BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 : G01F 15/06 | A1

(11) International Publication Number: WO 00/36380

(43) International Publication Date: 22 June 2000 (22.06.00)

(21) International Application Number: PCT/US99/29830

(22) International Filing Date: 15 December 1999 (15.12.99)

(30) Priority Data:
09/212,176 15 December 1998 (15.12.98) US

(63) Related by Continuation (CON) or Continuation-in-Part (CIP) to Earlier Application
US 09/212,176 (CIP)
Filed on 15 December 1998 (15.12.98)

(71) Applicant *(for all designated States except US)*: DANIEL INDUSTRIES, INC. [US/US]; 9753 Pine Lake Drive, Houston, TX 77055 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: LAMBERT, J., David, A., [GB/US]; 24703 Lakebriar Drive, Katy, TX 77494 (US). MALIK, Vipin [IN/US]; 17202 Hamilwood Drive, Houston, TX 77095 (US). HEUER, Rick [US/US]; 21315 Park Orchard, Katy, TX 77450 (US).

(74) Agents: ROSE, David, A. et al.; Conley, Rose & Tayon, p.c., P.O. Box 3267, Houston, TX 77253-3267 (US).

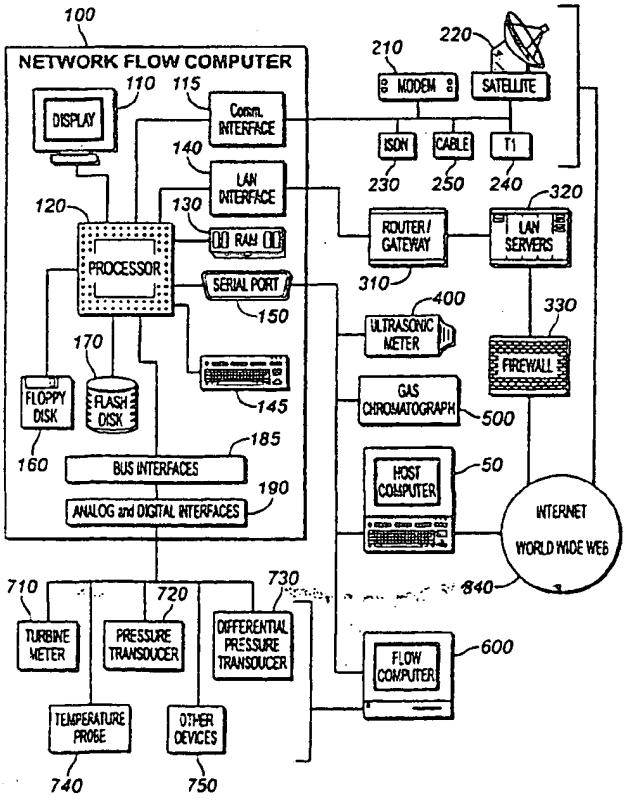
(81) Designated States: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
With international search report.

(54) Title: INTERNET ENABLED NETWORK FLOW COMPUTER SYSTEM

(57) Abstract

An improved network flow system includes an Internet/intranet enabled network flow computer with at least one flow measurement device and a host computer capable of communicating using Internet technology or through local connection to the flow computer. In one embodiment, the Internet enabled network flow computer receives flow measurement data, in raw or calculated form, from one or more measurement devices. The host computer connects to the network flow computer via the Internet, an intranet or local connections. The host computer transmits or receives data from the network flow computer, including the capability to view flow data results in web page format on the host computer and the capability to remotely or locally configure and control the flow computer from the host computer.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

INTERNET ENABLED NETWORK FLOW COMPUTER SYSTEM

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims the benefit of U.S. Application Serial No. 09/212,176 filed December 15, 1998, and entitled Internet Enabled Network Flow Computer System, which is hereby incorporated by reference for all purposes.

STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT

Not applicable.

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention relates generally to computer systems for measuring and controlling the flow of liquids and gases through pipeline or conduit. More particularly, the present invention relates to flow computers that can be accessed remotely and locally. Still more particularly, the present invention relates to flow computers that function as web servers to enable an operator to control the flow computer, and obtain flow measurements from the flow computer via the Internet, intranet and similar technology.

Background of the Invention

Flow computers are used throughout the world to control and measure the flow of liquids and gases in pipelines. Flow computers are used in various industries such as in the oil and gas industry and in the water treatment industry. The flow computers, typically connected to a number of flow sensors, receive signals that indicate the flow of liquid or gas. The flow computer processes these signals to obtain desired flow calculations. In some industries, these flow calculations are used to determine the quantity of a commodity, such as oil or gas, that has passed through a pipeline. In some instances, the flow calculations are used as a basis for a business transaction to determine the volume of a commodity that has changed possession from one party to another. Because of the quantities and money involved, it is critical that the flow computer be extremely precise and extremely reliable, regardless of the environment in which it is used. In addition to this critical role of providing exact flow measurements, the flow computer also may be used to automatically control various pipeline equipment such as valves, meters, electronic switches and other devices for controlling the operation and flow of the liquid or gas.

As shown in Figure 1, a conventional flow computer 10 includes a processor 11 with associated secondary storage device 13, keypad 14, LCD display 15 and data ports 17-23. Flow computer 10 may also include a floppy drive 16. Connected to flow computer 10 by data ports 17-

23 may be various flow measurement transducers 24-26, control devices 29-30, modem 27, and serial input/output device 28, such as a host computer 31. Data ports 17-23 may be RS-232 or RS-485 data ports.

Microprocessor (or "processor") 11 operates as the "brains" of the flow computer 10. Microprocessor 11 connects to a pipeline, for example, by flow measurement transducers 24-26. A variety of appropriate transducers 24-26 are known that provide data in analog voltage, analog current, frequency or other formats but nonetheless all of these transducers provide monitoring data regarding the amount and character of the fluid flowing through the pipeline. Microprocessor 11 monitors the data from the transducers, performs calculations, and carries out other functions. Software is embedded in the secondary storage device 13 that allows the microprocessor 11 to perform these functions. The secondary storage device 13 typically is an expensive solid state device known as a "flash disk" or ROM chip, which can withstand the harsh environments and extreme weather conditions in which flow computers are often used.

The flow computer 10 may be controlled by an "on-site" operator. Keyboard 14 allows an operator to input information such as commands that determine the frequency with which flow measurement readings are taken and results calculated. An operator may also choose to use the floppy drive 16 or other appropriate memory device to load and store data into the flow computer 10. Processor 11 provides a visual feedback to an operator by LCD display screen 15. Alternately, an operator may choose to "plug in" a laptop computer or other interface device 31 to the flow computer via one of the data ports 17-23. This allows the owner or operator of the flow computer 10 to easily obtain flow measurement information from the flow computer, and also to reprogram or modify the software of the flow computer when necessary.

Still referring to Figure 1, while it is possible to configure the flow computer 10 directly or locally, it is also known to communicate with the flow computer 10 remotely. For example, modem 27 connects to a phone line 32 and allows remote connection of a host computer to the flow computer 10. When flow computers are located in hard to access locations, a satellite antenna device may be used to form a satellite link to transmit information to and from a host computer. Thus, if an oil company desires flow data from a deepwater offshore platform, the oil company can establish communication with the flow computer from a host computer via a satellite connection, and then obtain the desired flow information. Alternatively, an operator may travel to the offshore platform and connect a laptop to the flow computer to retrieve the desired information. Remote communication with the flow computer provides the advantages of an on-site link with the flow

computer via a laptop computer without needing to be physically connected to the flow computer. However, several problems with this approach exist in present systems.

While this system permits flow measurements to be readily obtained at a remote location, several problems exist. The first problem is that manufacturers of flow computers usually implement their own proprietary communication protocol in their flow computers 10. The manufacturers may even use a customized processor that uses a proprietary operating system. Thus, the host computer 31 must have specialized software that is capable of interfacing and communicating with the proprietary software running on the flow computer in order to configure or retrieve information from the flow computer. Similarly, if a laptop is used to communicate directly with the flow computer, the laptop must use the appropriate proprietary communication protocol. As one of ordinary skill will appreciate, customers may employ flow computers from several manufacturers. In this event, the customer must keep track of the manufacturer of each flow computer so that the proper communication protocol can be used when communicating with a particular flow computer.

Further complicating this is that flow computer manufacturers periodically release new versions of software for their flow computers. Upon the flow computer being updated with new software, the host computer or laptop computer must also be updated. Consequently, the customer must record not only the manufacturer, but also the version of software that resides in the flow computer to ensure that proper communication and control can be exercised from the host computer or laptop computer. Thus, each time a new feature is added to the software or a new or modified protocol is used, the host must be correspondingly modified/updated. Inevitably there exists numerous versions of software running on different hosts. As a result, a significant problem arises in that a particular host machine, depending on which version of host software it is running, may or may not be able to effectively communicate and access some or all features of a given flow computer. In addition, the manufacturer of the flow computer is faced with the enormous task of attempting to ensure backward compatibility with host software that has not yet been updated to the latest software version.

The software version compatibility problems cause user confusion in the field and customer service nightmares. This problem has existed for years without any viable solution. Despite this well recognized problem, no effective solution has appeared.

In addition to the absence of an industry standard protocol, an industry standard protocol may be too limiting as technology advances. MODBUS, which currently is an industry standard bus that is used to connect flow transducers such as 24-26 to a flow computer 10, is one example.

The MODBUS technology is approximately two decades old and has failed to keep pace with the many technological advancements that have occurred over the years. Therefore, in order to make MODBUS work with some current technology, changes must be made to MODBUS. The problem is that once the standard MODBUS architecture is modified, then incompatibilities are introduced between the flow computer and the transducers unless both are using the same version of MODBUS. Thus, unless an industry standard architecture or protocol is continually updated, the standard becomes outdated.

It would be desirable if a standardized architecture and protocol could be developed for flow computers and host computers, while allowing sufficient flexibility to permit technological improvements to be implemented in flow computer technology. Despite the apparent advantages such a system offers, there has been no solution to these problems.

BRIEF SUMMARY OF THE INVENTION

An embodiment of the invention comprises a network flow system including at least one measurement device, a flow computer configured as a web server and associated with the measurement device, the flow computer configured to receive data from it, and a host computer communicating with the flow computer either locally or via an Internet/intranet connection. Uniform resource locator ("URL") address may therefore identify the flow computer, and the host may receive data from the flow computer in a web page format. The flow computer may also self-configure to identify and communicate with an attached measurement device. The flow computer may have a scalable operating system compressed onto a flash disk, and may be remotely programmed by a host computer. The host computer may communicate with the flow computer by a web browser.

Another embodiment of the invention is a method to obtain measurement data. The method includes coupling a flow computer to a measurement device suitable to monitor a fluid and provide corresponding measurement data, and connecting the flow computer via a communication link to a host computer, the flow computer obtaining measurement data from the measurement device using a first protocol and communicating with the host computer using a second protocol, the second protocol being suitable for Internet/intranet communication. Other aspects of this method may also be present. For example, the measurement device may be queried at regular intervals, the host computer may program the flow computer, the flow computer may be identified using a URL address, and the flow computer may generate a web page including data from the measurement device.

BRIEF DESCRIPTION OF THE DRAWINGS

A better understanding of the present invention can be obtained when the following detailed description of the preferred embodiment is considered in conjunction with the following drawings:

Figure 1 is a block diagram of a prior art flow computer system;

Figure 2 is a block diagram of an improved flow computer and host system constructed in accordance with the preferred embodiment;

Figure 3 is a block diagram of the network flow computer of Figure 2 constructed in accordance with the principles of the present invention;

Figure 4 is a block diagram of the software implemented in the network flow computer of Figure 2;

Figure 5 is a block diagram of the flash disk, BIOS and RAM disk of the network flow computer of Figure 2; and

Figure 6 is a flow chart of the flash disk initialization routine performed on the network flow computer of Figure 2.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The following detailed description describes the preferred embodiment for implementing the underlying principles of the present invention. One skilled in the art should understand, however, that the following description is meant to be illustrative of the present invention, and should not be construed as limiting the principles discussed herein. In addition, certain terms are used throughout the following description and claims to refer to particular system components. As one skilled in the art will appreciate, manufacturers may refer to a component by different names. This document does not intend to distinguish between components that differ in name but not function. In the following discussion and in the claims, the terms "including" and "comprising" are used in an open-ended fashion, and thus should be interpreted to mean "includes, but is not limited to ...". Additionally, the term "fluid" is used in a manner including liquids and gases. Also, the term "couple" or "connect" is intended to mean either an indirect or direct electrical connection. Thus, if a first device couples or connects to a second device, that connection may be through a direct electrical connection, or through an indirect electrical connection via other devices and connections.

Referring now to Figure 2, a host computer 50 couples to one or more network flow computers 100, 800, 900 via the Internet, intranet or similar technology. The flow computers also connect to various flow measurement devices 710, 720, 730 via a standardized bus, such as a MODBUS. Although three such measurement devices have been shown connected to each of the

flow computers in Figure 2, one skilled in the art will understand that the number of flow measurement devices and other equipment may vary. Similarly, the host computer 50 may act as a host to an unspecified number of flow computers, without limitation.

According to the preferred embodiment of the present invention, the flow computer is programmed to collect or receive data from a measurement device such as 710, 720, 730. Flow computer 100 is also programmed to be a web server, such that it can communicate over an Internet, intranet or similar technology connection. In addition to operating as a web server, the network flow computer 100 also executes other flow computing software.

The host computer 50 may comprise any PC (personal computer), laptop computer, work station, mainframe computer, or any other computer capable of supporting Internet, intranet or similar technology access. The network flow computer 100 may be based on the INTEL *x*86 family of microprocessors or compatible processors, such as those manufactured by AMD in its K6 and K7 family, CYRIX's PR series, and IDT's C6 and C7 family of microprocessors. The network flow computer 100 may also be based on SUN MICROSYSTEM's RS 6000, SPARC, or ALPHA RISC microprocessor based machines. Alternatively, the network flow computer 100 may be based on any other microprocessor that is capable of performing the operations required in the present invention. The preferred embodiment contemplates use of an INTEL *x*86 based computer as the most cost-efficient platform. The minimum system requirement for the network flow computer on an *x*86 platform is a 386 class microprocessor, such as that commercially available from manufacturers like INTEL, AMD and CYRIX. The invention will also work on higher class *x*86 architecture microprocessors such as the 486 family, the PENTIUM, the PENTIUM PRO, the PENTIUM II, and the like. As explained above, the invention will also work on non-*x*86 platform computers. While the best mode implements a standard PC configuration, the invention can also be implemented in a machine with an embedded computer or processor.

The host computer 50 preferably runs an Internet web browser software package, such as MICROSOFT's INTERNET EXPLORER or NETSCAPE's NAVIGATOR software, although any generic browser may be used. No other specialized software is required at the host computer. However, if desired, the host computer 50 may be programmed with specialized software to enable backward compatibility with older versions of software on the network flow computer. Such specialized software can co-exist with the software implemented in the preferred embodiment of the present invention.

The host computer 50 may connect to the Internet directly or through a service provider. Similarly, both host computer 50 and network flow computer 100 may connect to the Internet using

any communication media, including a standard analog phone line connection, an ISDN connection, a T1 connection, a satellite connection, direct wire connection, or the like. Host computer 50 may also be directly connected to flow computer 100, bypassing the public Internet, such as through the use of intranet and similar technology.

During operation, flow measurement devices 710, 720 and 730 monitor and measure certain parameters of a fluid or fluids of interest. Flow computer 100 receives flow measurements from the flow measurement devices 710, 720 and 730 and performs flow calculations. The flow computer then displays these flow calculations on a web page real-time. The flow computer 100 may also be requested to display historical and other measurement or configuration data. Each flow computer therefore has its own web address. Host computer 50 accesses these flow calculations by simply entering the respective URL address for a particular flow computer. For example, to address flow computer 100, an operator of host computer 50 selects the Internet URL address for flow computer 100, which as shown in the example of Figure 2 is www.flow#1.com. While the present invention refers to URL to identify the address of flow computer 100, the present invention encompasses any other type of addressing that may be used to identify a flow computer. In response to the selection of this address, flow computer 100 is contacted and replies with its web page, which preferably includes a list of current flow calculations. Other features and details of the flow computer 100 may be selected from menu entries in that computer's web page. In similar fashion, each of the other flow computers in the customer's inventory may be addressed and its information monitored real-time via the Internet, intranet or similar technology.

The configuration shown in Figure 2 solves many of the problems associated with the prior art. Prior art measurement devices 710, 720, 730 may operate using only a single industry standard protocol. For example, gas chromatographs use only the MODBUS protocol to communicate with external devices. When a gas chromatograph connects to a network flow computer using industry standard MODBUS protocol, the gas chromatograph becomes, in effect, a networked gas chromatograph that is accessible remotely via the network flow computer. Additionally, the software on the gas chromatograph does not require an upgrade. So long as the gas chromatograph is coupled to the network flow computer 100 as explained above, the network flow computer can make the flow measurement data provided by the gas chromatograph accessible remotely by any host computer 50 located anywhere in the world. This eliminates the need to ensure that the flow computer and host computer are running the same version of software. Further, this solution is particularly practical in the real world because of the wide availability of Internet connections.

The principles of the present invention may be applied to existing meters, including gas chromotographs, ultrasonic meters, and orifice meters. For example, these meters may already include in their design associated electronics. These electronics may be integrated with a network flow computer so that such meters in effect become Internet enabled meters with a builtin network flow computer. When the present invention is used in such meters, the meters are capable of functioning in a manner similar to the preferred embodiment of the present invention in addition to all of the existing functionality within the meters.

Referring now to Figure 3, a network flow computer 100 constructed in accordance with a preferred embodiment includes a PENTIUM family microprocessor operating at a clock speed of 100Mhz or 133Mhz. The network flow computer 100 preferably includes a minimum of 2 megabytes of flash disk 170, with 5-10 MB being most preferred. The actual size of the flash disk is somewhat dependent on the number of options or features incorporated into the system. The flow computer 100 also preferably includes at least 8 megabytes of RAM 130, with 16-32 megabytes being most preferred. The flow computer 100 also includes a display mechanism 110. The display 110 may be a standalone monitor in a flow computer implemented in a standard PC configuration. When the flow computer 100 is implemented in a machine with an embedded computer or processor, the display 110 may be an LCD display unit such as a 2" x 16" text-only LCD display. The flow computer 100 also includes a keyboard 145 and may include floppy disk 160.

The network flow computer 100 also preferably includes one or more bus interfaces 185, such as ISA and PCI bus interfaces. Analog and digital interface cards 190 in the flow computer 100 are coupled to the bus interfaces 185 of the flow computer. Flow measurement devices such as turbine meter 710, pressure transducer 720, differential pressure transducer 730, temperature probe 740 and other flow measurement devices 750 are coupled to the analog or digital interface cards 190. Such flow measurement devices provide flow data to the flow computer 100 on a real-time basis. When flow measurement devices 710, 720, 730, 740 and 750 are directly connected to network flow computer 100, the network flow computer becomes the flow meter for the flow measurement devices. As a flow meter, the network flow computer 100 receives and processes digital and analog signals sent by the flow measurement devices 710, 720, 730, 740 and 750. The network flow computer 100 then performs all necessary calculations and corrections to the raw data received from the flow measurement devices 710, 720, 730, 740 and 750. The flow computer 100 stores the calculated flow measurement results in log files on the flash disk 170.

The flow computer may also self-configure to identify and communicate with an attached measurement device. As one of ordinary skill in the art will acknowledge and appreciate, methods of identifying peripheral devices attached to a computer are well known in the industry.

The host computer 50 may connect to the network flow computer 100 via the Internet 340 to view the flow results real-time. As previously stated, host computer 50 may also connect to the network flow computer 100 via intranet or similar technology. On the host computer 50, the calculated flow results are displayed as graphs, tables of data or in any other desired manner of display. In addition, the flow computer 100 may store some or all of the calculated flow results for subsequent analysis by the host computer 50.

Still referring to Figure 3, the network flow computer 100 also preferably includes one or more serial ports 150, such as RS-232 or RS-485 serial ports. The serial ports 150 enable the network flow computer 100 to communicate with various external devices. The network flow computer 100 is coupled via serial ports 150 to ultrasonic meter 400 and gas chromatograph 500. The industry standard MODBUS protocol is used by the flow computer 100 to communicate via the RS-232 or RS-485 serial ports 150 with the ultrasonic meter 400, gas chromatograph 500, and flow computer 600. The serial port 150 is also used to provide direct connection of the host computer 50 to the network flow computer 100. Direct connection of the host computer 50 to the network flow computer 100 via the serial port 150 is oftentimes necessary when the host computer user is physically located at the network flow computer's site for various reasons such as when troubleshooting or servicing the network flow computer. Gas chromatograph 500 is connected to the network flow computer 100 via RS-232 or RS-485 serial ports 150 using industry standard MODBUS protocol.

Still referring to Figure 3, the network flow computer 100 is coupled via serial port 150 to the ultrasonic meter 400. An ultrasonic meter 400 may also be connected to the network flow computer 100 via the LAN interface 140 and LAN servers 320. The ultrasonic meter 400 may be strictly a flow meter or a flow computer. As a flow meter, the ultrasonic meter 400 records raw velocity flow data and does not perform any data corrections such as for temperature and pressure. The raw data is then communicated by the ultrasonic meter 400 to the flow computer 100 which performs all necessary calculations, including correcting the data for temperature and pressure. As a flow computer, the ultrasonic meter 400 not only records the raw data but also performs the necessary calculations, corrects the data for temperature and pressure, and finally communicates the calculated results to the network flow computer 100.

The network flow computer 100 is coupled to other flow computers via serial ports 150. For example, network flow computer 100 may be coupled to another flow computer 600, such as the commercially available flow computer DANIEL SPECTRA 100, via the serial port 150. The external flow computer 600 may itself be coupled to other flow measurement devices such as devices 710, 720, 730, 740 and 750. In such a configuration, the network flow computer 100 is indirectly connected to flow measurement devices 710, 720, 730, 740 and 750 via flow computer 600. In an indirect connection of the network flow computer 100 to flow measurement devices 710, 720, 730, 740 and 750, the flow computer 600 receives and processes the digital and analog flow signals from the measurement devices and typically the flow computer 600 performs all necessary flow measurement calculations. The computed results are then communicated by the flow computer 600 to the network flow computer 100. After receiving the computed results, the network flow computer 100 stores the results in log files on flash disk 170.

In the preferred embodiment of the present invention, the network flow computer 100 connects to a host computer 50 via the Internet and the World Wide Web 340. As previously stated and as one of ordinary skill in the art will appreciate, network flow computer 100 may also be connected to host computer 50 via an intranet and similar technology. The flow computer 100 may connect to the Internet and the World Wide Web 340 in many different ways. In the preferred embodiment, the flow computer 100 connects to the Internet and the World Wide Web 340 via a local area network ("LAN"). The network flow computer 100 includes a LAN interface 140. While the preferred embodiment uses the industry standard Ethernet LAN, one skilled in the art of the present invention will appreciate that other commercially available LANs such as LON WORKS, PROFI BUS, DEVICE NET, CAN, USB, or any other industry standard LAN will also be appropriate. The LAN connection 140 enables connection of the flow computer 100 to the Internet and the World Wide Web 340. The LAN connection 140 also enables the network flow computer 100 to be networked to other equipment at the site where the flow computer is being used. To enable the connection of the flow computer to the Internet 340, the LAN interface 140 is coupled to a router or gateway 310, which in turn is coupled to LAN servers 320. To prevent unauthorized access, infiltration of the LAN by viruses from the Internet, and for other security reasons, the LAN connection 140 of the flow computer 100 is secured by coupling the LAN servers 320 to a firewall 330. The firewall 330 is then coupled to the Internet and World Wide Web 340. While in the preferred embodiment the flow computer 100 is coupled to the Internet and World Wide Web 340, one skilled in the art will appreciate that the present invention will similarly work

with any future improvements to the Internet, the World Wide Web or any other aspect of the Internet, including what is commonly referred to as the forthcoming "Internet II."

Still referring to Figure 3, the network flow computer 100 may also be connected to the Internet and the World Wide Web 340 via other communication media. Therefore, the flow computer 100 also preferably includes one or more communication interface cards 115 coupled to various external communication devices via modem 210, satellite 220, ISDN 230, T1 line 240 and cable 250. Any one or more of the communication media, modem 210, satellite 220, ISDN 230, T1 line 240, and cable 250, may be used to connect the network flow computer 100 to the Internet and World Wide Web 340. Similarly, the host computer 50 is connected to the Internet and World Wide Web 340.

Referring now to Figure 4, custom software has been written and combined with off-the-shelf commercial or publicly available software to implement the network flow computer 100. The combination, order, layering or hierarchy of software in Figure 4 is intended to simplify discussion of software used in the preferred embodiment. It should not be interpreted as a limitation of the present invention. Different combination, order, layering or hierarchy of software may be used in the present invention. Software used in the preferred embodiment of the present invention includes the LINUX operating system ("OS") 101, network services software NET SERVICES 102, industry standard MODBUS communication protocol 103, the APACHE web server 104, CGI scripts and HTML code 105, and JAVA applets and C programs 106.

The base platform for the software in the network flow computer 100 preferably comprises RED HAT LINUX Operating System ("OS") 101. LINUX OS 101 is a multitasking and multi-user operating system. RED HAT LINUX OS 101 is a variant of the popular UNIX operating system. RED HAT LINUX OS is commercially available from RED HAT SOFTWARE, INC. (http://www.redhat.com). The LINUX OS 101 contains the software "kernel" for the network flow computer 100. As one skilled in the art will understand, the kernel is software code within the operating system that functions as the master controller of all system operations and resources. The kernel contains the base functionality of the operating system, but not the file system. The kernel is responsible for allocating computer resources. The kernel requires a file system to execute files, locate files, and perform other file operations. There are many different types of files, including data files and code files. Examples of code files are EMACS, a text editor, and X-WINDOWS, a graphical shell for LINUX OS 101.

While the preferred embodiment of the present invention implements the RED HAT LINUX OS 101, other distributions of LINUX or other operating systems may be used if desired.

Examples of other publicly available distributions of LINUX include SLACKWARE LINUX, YGGDRASIL LINUX, and CALDERA LINUX. Although the present invention will work with full-featured operating systems, such as WINDOWS 98 or WINDOWS NT, it is preferable to use an operating system that is scalable, such as LINUX OS 101 or MICROSOFT WINDOWS CE. A scalable operating system permits selection of a subset of the operating system modules or features that are necessary for the application at hand. The memory space required by the subset of the operating system features or modules is controlled and can therefore be minimized. This capability of scalable operating systems is particularly desirable in development of embedded systems where memory space may be limited and expensive. For example, in the present invention LINUX OS 101 with a selected set of features requires only approximately 2 megabytes to boot, whereas WINDOWS 98 may require upwards of 100 megabytes. Furthermore, because the network flow computer 100 is preferably designed for harsh environments, relatively expensive memory components, such as flash disks 170, are used. Thus, the use of a scalable operating system, such as LINUX OS 101, is desired to minimize the amount of memory required in the flow computer, and consequently minimize the cost of the system.

Still referring to Figure 4, in the preferred embodiment, the size of the embedded LINUX OS 101 is reduced down to 1 to 2 megabytes of storage by recompiling the LINUX OS source code with a subset of the full features of the standard LINUX OS. This is accomplished by running a configuration utility provided by RED HAT LINUX. The utility enables selection of a subset of the functions provided in the standard LINUX installation. The utility is executed in the X-WINDOWS shell provided by LINUX. The command "make xconfig" is executed from within X-WINDOWS to initiate the utility. Once the utility is initiated, the options identified in attached Appendix A are selected. Once the identified LINUX OS options have been selected, the configuration utility is used to recompile the source code of LINUX OS 101 to generate a subset version of the OS where the size of the OS has been minimized to save storage space on the flash disk 170 of the network flow computer 100.

The next "layer" of software in the network flow computer 100 is the NET SERVICES 102 bundle of software. The NET SERVICES software 102 is a combination of one or more system programs (as opposed to application programs), that provide various system services or resources and work in conjunction with the LINUX OS 101. For example, the preferred embodiment uses NET SERVICES such as the FILE TRANSFER PROTOCOL ("FTP"), HYPER TEXT TRANSFER PROTOCOL ("HTTP") and TELNET. However, the present invention may use other NET SERVICES 102. NET SERVICES 102 programs are publicly available and may be

acquired from many different sources. For example, as in the preferred embodiment, the NET SERVICES 102 software may be acquired from the RED HAT LINUX distribution CD-ROM. Most publicly available LINUX distributions typically include NET SERVICES 102 software. As previously mentioned, examples of other publicly available LINUX distributions include SLACKWARE LINUX, YGGDRASIL LINUX, and CALDERA LINUX. NET SERVICES 102 software may also be readily downloaded from various Internet sites. Such Internet sites are easily identified using commonly available Internet search engines.

Still referring to Figure 4, the preferred embodiment uses FTP net service 102 to transfer files between the host computer 50 and the network flow computer 100. The HTTP net service 102 enables the network flow computer 100 to communicate in the World Wide Web of the Internet. Consequently, an operator may view flow measurement results from the network flow computer 100 via a host computer 50 using an Internet browser, such as MICROSOFT INTERNET EXPLORER or NETSCAPE NAVIGATOR. The preferred embodiment uses TELNET net service 102 to enable an operator to remotely login to the network flow computer 100 as a terminal. Such a remote connection appears to the network flow computer 100 as if the technician is physically located where the network flow computer resides and permits the technician to work directly on the network flow computer. Furthermore, such a connection enables the technician to troubleshoot the network flow computer 100 from a remote site.

The MODBUS software 103 enables the network flow computer 100 to communicate via the RS-232 or RS-485 serial ports 150 with legacy flow computers or flow meters such as the ultrasonic meter 400, gas chromatograph 500, and other flow computers 600. MODBUS software 103 implements the standard MODBUS communication protocol to work in conjunction with LINUX OS 101. The source code in Appendix B is an implementation of the MODBUS protocol using the C programming language.

The APACHE Web Server software 104 enables the network flow computer 100 to act as web server using HTTP and FTP NET SERVICES 102 communication protocols. As a web server, the network flow computer 100 may be accessed via the host computer 50 over the Internet 340. Similarly, the network flow computer 100 may be accessed via the host computer 50 over an intranet or similar technology. The APACHE Web Server software 104 also enables use of COMMON GATEWAY INTERFACE ("CGI") scripts 105, discussed below. The APACHE WEB SERVER software 104 is a publicly available web server software and may be easily downloaded from the Internet. Internet sites for downloading the APACHE Web Server software 104 may be

identified by using commonly available Internet search engines. A prominent Internet web site for downloading the APACHE WEB SERVER software 104 is www.apache.org.

Still referring to Figure 4, the CGI scripts and HTML code 105, and JAVA applets and C programs 106, are application programs that enable the network flow computer 100 to perform various tasks as part of its overall function of communicating with various external devices and presenting calculated flow data results to host computers 50 connected to the network flow computer via the Internet 340. As one skilled in the art of the present invention will appreciate, various such programs may be written to enable the network flow computer 100 to perform various functions. For example, the source code provided in Appendix C enables the network flow computer 100 to query the ultrasonic meter 400 once every sixty seconds to obtain corrected flow rates. Once obtained, the flow computer 100 appends the flow rates to a log file named "ultrasonic.log." One such file is created for each full day and the process is repeated every twenty-four hours. Similarly, Appendix D is an example which lists source code enabling the flow computer 100 to read predefined message blocks from the ultrasonic meter 400. The message blocks contain data used to populate tables. The tables are then displayed on a host computer 50 connected to the flow computer 100 via the Internet 340. Both of the foregoing examples also use the MODBUS protocol 103 programs listed in Appendix B.

The CGI scripts and HTML code 105 use JAVA applets to display the computer flow data results in a graphical format on the host computer 50 when the host is connected to the flow computer 100 via the Internet 340. The JAVA applets are commercially available under the brand name JAVACHART from VISUAL ENGINEERING, INC. As one skilled in the art of the present invention will appreciate, the functionality provided by the commercial software JAVACHART can be independently developed using JAVA programming.

Referring now to Figures 3 and 5, the flash disk 170 of the network flow computer 100 contains the software files identified in Figure 4. These programs and data execute when the network flow computer 100 is used for its intended purpose. However, the flash disk 170 must be initialized before it can operate properly. Similar to the initial formatting and installation of an operating system and other software and data on the hard disk of a common PC, the initialization of flash disk 170 is typically performed only once per unit. However, the initialization process may be repeated as necessary. Prior to initialization, the flash disk 170 is either unformatted or contains files which must be deleted.

Referring now to Figure 5, as previously discussed, the network flow computer 100 includes flash disk 170 and RAM 130. The network flow computer 100 also includes BIOS 121.

Flash disk 170 includes LINUX LILO loader 171 on the master boot record ("MBR") of the flash disk. Flash disk 170 also includes non-compressed file space 172, compressed file space 173, user data area 174 and spare space 175. RAM 130 includes RAM Disk 131, a non-compressed file space.

In a regular PC, the file system is generally not compressed and resides on the hard disk of the PC. The hard disk in a PC is a relatively inexpensive storage medium. Therefore, it is economically justifiable to have non-compressed file systems for general software applications on a regular PC. Generally, to run software on a standard PC configuration, the program gets dynamically loaded from the PC's hard disk to the PC's random access memory ("RAM") where the programs are executed. The file system in a standard PC configuration remains on the hard disk and programs read from and write to the file system on the hard disk. However, the programs must reside in RAM during execution. Programs cannot execute on the PC's hard disk.

The preferred embodiment of the present invention does not use hard disks as a form of non-volatile permanent storage medium. This is because present day hard drive technology generally cannot withstand the harsh environments in which flow computers may be used. Instead, the preferred embodiment uses flash disk 170 for non-volatile permanent storage. The flash disk is necessary because files must be stored in some permanent non-volatile storage medium. A non-volatile storage medium does not lose data stored on it when electrical power to the storage medium is terminated. A flash disk emulates a physical magnetic hard drive. Therefore, software running on the network flow computer 100, cannot distinguish between hard disks and flash disks. Flash disks use solid state technology with no moving parts. Therefore, flash disks are better able to withstand adverse environmental conditions such as shock and temperature highs and lows. Flash disks have the added advantage of smaller physical size compared to hard disks. This facilitates use of flash disks in network flow computers configured as embedded computers or processors.

Flash disks are very expensive compared to the cost of hard disks. In a regular PC with hard disk, volatile RAM storage costs are approximately 20 to 30 times higher than the non-volatile hard disk storage. Since the inexpensive storage medium in a regular PC is the hard disk, a non-compressed hard disk is an acceptable mode of operation from a cost perspective. The situation is reversed in the network flow computer 100 of the preferred embodiment. In network flow computer 100, the non-volatile permanent flash disk 170 storage medium is approximately five times more expensive than the volatile temporary RAM 130 storage. Since the flash disk 170 of the network flow computer 100 is more expensive than RAM 130, in order

to minimize the overall cost of the flow computer it is highly desirable to use more RAM storage and less flash disk storage. However, the foregoing desirability is constrained by the technological limitation preventing permanent storage of files on RAM 130. Therefore, to reduce cost, the preferred embodiment minimizes flash disk 170 storage by storing most of the files on the flash disk in compressed form in the compressed file space 173. When the network flow computer 100 runs, it transfers the compressed files 173 from the flash disk 170 in a non-compressed form to the RAM 130. Software on the flow computer 100 is executed by accessing the non-compressed file space 131 in RAM and not by accessing the compressed file space 173 in the flash disk 170.

Referring now to Figures 4 and 5, when initialization of flash disk 170 is complete the flash disk contains at least some of the software files identified in Figure 4, in addition to other possible data and program files. More particularly, the LINUX LILO loader 171 resides, starting at sector zero, on the MBR of the flash disk 170. In between the memory space immediately following MBR 171 and sector 430, resides the non-compressed file space 172. The non-compressed file space 172 contains the non-compressed file system, the LINUX OS 101 kernel and other files. In the space immediately following sector 430, resides the compressed file space 173. The compressed file space 173 contains software and files identified in Figure 4 which are not loaded in the non-compressed file space 172. The user data area 174 preferably contains data files used by the flow computer 100 that may change during the use of the flow computer. Depending on the size of flash disk 170, there may be a portion of the flash disk that is spare space 175. The particulars of the flash disk 170 initialization routine are described later in this disclosure.

User data area 174 of flash disk 170 contains data files which may change during the use of network flow computer 100. Example of such user data files includes files which store configuration information about the flow computer. Configuration files are modified when, for example, host computer 50 connects to network flow computer 100 and changes the frequency of flow measurements from every four hours to every six hours. In order to permanently save changes made to user data files, such files are stored on the flash disk 170 in user data area 174 and any changes to the files are contemporaneously recorded on the flash disk and not in RAM 130. The user data area 174 may be compressed or uncompressed depending on various considerations such as performance needs, available flash disk space and other considerations.

The flash disk 170 storage space needed for LILO loader 171 and non-compressed file space 172 is determined by pre-compiling the LINUX OS 101 kernel. In the preferred

embodiment, 430 sectors is necessary to store LILO loader 171 and non-compressed file space 172 on the flash disk 170. As previously discussed, the size of LINUX OS 101 and the kernel may vary depending on the particular subset of the full features selected from the standard LINUX OS. Furthermore, the size may also be influenced by any customized changes made to the LINUX OS 101 kernel.

Still referring to Figure 5, The LINUX OS 101 kernel is loaded in the non-compressed file space 172. A small segment of the kernel is stored in non-compressed form in non-compressed form in non-compressed file space 172. Most of the kernel is stored in compressed form in non-compressed file space 172. When network flow computer 100 is powered on or re-booted, the non-compressed segment of the kernel in non-compressed file space 172 is executed first. Once execution of the non-compressed kernel begins, the kernel decompresses and executes other compressed segments of the kernel in non-compressed file space 172 as needed.

In the preferred embodiment the non-compressed file space 131 in RAM is made available by creating a RAM disk. A RAM disk is the method of creating a "hard disk" or "flash disk" in RAM to emulate the functionality of a hard disk or flash disk. Therefore, RAM disk 131 can store a file system and files may be written to and accessed from the RAM disk just like a conventional PC hard disk or flash disk. From the perspective of the software (Figure 4) running on the network flow computer 100, a file system which resides on RAM disk 131 is indistinguishable from a file system which resides on flash disk 170. Only the LINUX OS 101 kernel can distinguish among a RAM disk 131 and a hard disk. The ability to create a RAM disk is native to the LINUX OS 101 Kernel.

While the creation and use of the compressed file space 173 and RAM disk 131 is desirable from cost-savings perspective, it is not a requirement of the present invention that a compressed file space and RAM disk be used in the network flow computer 100. The need to have a compressed file space 173 and RAM disk 131 is a cost-effectiveness factor. The intended invention will work with flash disk 170 which has only non-compressed file space 172. Similarly, the intended invention will work with RAM 130 which does not have a RAM disk 131. If in the future the cost of flash disk 170 becomes less than the cost of RAM 130, then the use of compressed file space 173 and RAM disk 131 may be unnecessary and the related cost-savings may become a non-factor.

Upon power up or reset of the network flow computer 100, the BIOS 121 is in control of the flow computer. The BIOS 121 is an integrated circuit chip which resides on the flow computer 100 motherboard. The BIOS 121 is preprogrammed with logic which boots (i.e., takes

control of) the flow computer 100 when it is initially powered on or reset. When the BIOS 121 begins the booting process, it searches for the hard drive or flash disk, whichever is connected as the first drive or "hda" as it is referred to in LINUX OS terminology. The BIOS 121 then searches for the MBR on "hda." Since the hard disk is initially installed as "hda," the BIOS locates the MBR of the hard disk. The LINUX OS boot loader installed on the MBR of the hard disk boots the network flow computer 100 during the initialization of the flash disk 170.

Still referring to Figure 5, the LILO loader 171 must be located on the MBR of flash disk 170 because this is where BIOS 121 will search to initiate the boot sequence for the network flow computer 100. After the BIOS 121 finds and begins execution of the LILO loader 171, the LILO loader searches and finds the LINUX OS 101 kernel in the non-compressed file space 172. Once the LINUX OS 101 kernel is found, LILO loader 171 loads the LINUX OS kernel in RAM 130 and begins executing the kernel. At this point the LINUX OS 101 kernel controls the boot sequence of the network flow computer 100.

The LINUX OS 101 kernel may be physically stored anywhere on the flash disk 170. However, in order for the LILO loader 171 to find the kernel, the loader must either be specifically instructed as to the physical location of the kernel or the loader must find the kernel in the default physical location. The default physical location for the kernel is the storage space beginning immediately following the storage space occupied by the LILO loader 171 on the flash disk 170. In the preferred embodiment, the LINUX OS 101 kernel is physically stored at the beginning of the non-compressed file space 172 on the flash disk 170. This is the default location for the LINUX OS 101 kernel.

The LINUX OS 101 kernel has primary responsibility for booting the network flow computer 100. To properly execute the boot sequence, the kernel determines the location of the file system, whether the file system is compressed and whether a RAM disk is created. If a RAM disk is created, the kernel determines the size of the RAM disk and whether the file system is loaded into the RAM disk from the flash disk. The kernel makes the foregoing decisions based on command line parameters passed from the LILO loader 171 to the LINUX OS 101 kernel.

As previously discussed, conventionally the file system resides on the hard disk or flash disk and programs access and write files from and to the hard disk or flash disk. When a RAM disk is setup and the file system from the flash disk is transferred to the RAM disk, programs running on the network flow computer 100 access and write files from and to the RAM disk as opposed to the flash disk.

In the preferred embodiment, the compressed file space 173 is stored on the flash disk 170 beginning at sector 430. This is the storage space immediately following the non-compressed file space 172 in the flash disk 170.

Still referring to Figure 5, the initialization of flash disk 170 will now be described in accordance with the preferred embodiment. Other initialization sequences may also be appropriate, and the following discusses only the preferred initialization. As a prerequisite to the preferred initialization process, the network flow computer 100 must have a conventional hard disk installed as its first drive, "hda." The flash disk 170 must be initially installed in the network flow computer 100 as its second drive, "hdc" ("hdb" may be used instead). Additionally, the hard disk "hda" must have LINUX OS installed on it. This may be accomplished with commercially and publicly available off-shelf software package called RED HAT LINUX which provides detailed instructions on how to install LINUX OS onto the hard disk using the CD-ROM disk provided by RED HAT. Furthermore, all files and the associated directory structure that will eventually reside in the compressed file space 173 and user data area 174 of the flash disk 170 must be placed in a temporary directory on the local hard disk, "hda", of the network flow computer 100.

In the preferred embodiment, three program scripts, "makeLILObootdisk," "makeCompressedRamDisk" and "bdlilo.conf" listed in Appendix E, are used to implement the flash disk 170 initialization process. As one skilled in the art of the present invention will appreciate, the initialization process implemented in the three programs may be modified or written in many different variations and yet accomplish the same purpose.

The "makeLILObootdisk" and "makeCompressedRamDisk" programs (Appendix E) create the RAM disk 131 and the non-compressed file system within the RAM disk. The programs then copy a compressed version of the file space in RAM disk 131 to temporary storage on the hard disk connected as "hda." The programs next create the non-compressed file space 172 on the flash disk 170 "hdc" and copy the non-compressed and compressed segments of the LINUX OS 101 kernel and other files to the non-compressed file space 172. The programs then copy the boot loader file, "boot.b," from the boot directory of the hard disk "hda" to the boot directory in master boot record 171 of the flash disk 170 "hdc." The boot loader configuration file, "bdlilo.conf." is then copied to the flash disk 170. Next, programs and data files stored in temporary storage on hard disk "hda" must be copied to flash disk 170 "hdc" in compressed file space 173 and user data area 174 as appropriate. Other files, such as operating system files may also be copied into compressed file space 173.

The LILO boot loader configuration program, "bdlilo.conf" (Appendix E), is executed after the flash disk 170 initialization programs "makeLILObootdisk" and "makeCompressedRamDisk" have been successfully completed. The LILO boot loader program passes command line parameters to the LINUX OS 101 kernel in the non-compressed file space 172. The parameters inform the kernel that the network flow computer 100 will use a RAM disk 131, the size of the RAM disk, and the location of the compressed file space 173 on the flash disk 170. Additionally, the boot loader program also determines the particular kernel image to be used during the boot sequence for the network flow computer 100. In the preferred embodiment the kernel image used for the boot sequence is located in the file "vmlinuz.embedded."

The LILO boot loader configuration program, "bdlilo.conf" only writes to the first drive, "hda." However, as described above, during initialization of the flash disk 170, the first drive in the network flow computer 100 is the local hard disk "hda" and the flash disk is connected as the second drive, "hdc." If LILO boot loader program is permitted to write to "hda" during the initialization routine, the program will damage the LINUX OS that resides on the local hard disk "hda." Therefore, once initialization programs "makeLILObootdisk" and "makeCompressedRamDisk" have been successfully completed, the hard disk must be disconnected from the network flow computer 100 and the flash disk 170 must be connected as the first drive, "hda." However, at this point the initialization of the flash disk 170 is not yet complete and with the flash disk connected as "hda", the flow computer 100 cannot boot up using the flash disk. The solution is to create a LINUX OS bootable floppy disk. This can be accomplished by using the commercially and publicly available RED HAT LINUX OS CD-ROM distribution disk. With the bootable floppy disk in the floppy disk drive 160 of the network flow computer 100, the flow computer may be booted up with the flash disk 170 connected to the flow computer as the first drive, "hda." This configuration permits the connection of the flash disk 170 as the first drive, "hda," on the flow computer 100 during the LILO configuration phase of the flash disk initialization routine. Therefore, when the LILO loader program is executed, the program writes to "hda" where the flask disk 170 is connected. Thus, the desired result of configuring the flash disk is achieved. Once the LILO loader configuration is completed, the flash disk 170 initialization routine is complete and bootable floppy disk is removed from the floppy drive 160. The network flow computer 100 is now ready for use. When network flow computer 100 is re-booted, the flow computer boots up with the flash disk 170 connected as the first drive, "hda."

The foregoing initialization routine is presented in flow chart form in Figure 6. As one of ordinary skill in the art will recognize and appreciate, the foregoing initialization routine may be performed with variations from that described here without departing from the objective of the initialization routine.

The foregoing disclosure and description of the invention are illustrative and explanatory thereof, and various changes in the size, shape, materials, components, circuit elements, wiring connections and contacts, as well as in the details of the illustrated circuitry and construction and method of operation may be made without departing from the spirit of the invention.

Numerous variations and modifications will become apparent to those skilled in the art once the above disclosure is fully appreciated. It is intended that the following claims be interpreted to embrace all such variations and modifications.

PROGRAM CODE

APPENDICES A-E, Pages 22-69

APPENDIX A

LINUX OS OPTIONS SELECTED IN "makexconfig" CONFIGURATION UTILITY

1. In "Code Majority Level" options, "Yes" is selected for the following option:
   a. "Prompt for development and/or incomplete code drivers."
2. In "Loadable Module Support" options, "Yes" is selected for the following options:
   a. "Enable Loadable Module Support;"
   b. "Set Version Information on All Symbols For Modules;" and
   c. "Kernel daemon support."
3. In "General Setup" options, "Yes" is selected for the following options:
   a. "Kernel Math Emulation;"
   b. "Networking Support;" (this option enables TCPIP, a networking communication protocol);
   c. "PC Bus;"
   d. "PCI BIOS Support;"
   e. "System 5 IPC;"
   f. "Kernel Support for A.OUT Binaries;"
   g. "Kernel Support for F Binaries;"
   h. "Kernel Support for Java Binaries;" and
   i. "Compile Kernel as ELF if your GCC is ELF-GCC."
4. In response to "Processor Type," "486" is selected. This option designates the x86 family of microprocessors as the platform on which the network flow computer 100 will run.
5. In the "Floppy ID and Other Block Devices" options, "Yes" is selected for the following options:
   a. "Normal Floppy Disk Support;"
   b. "Enhanced IDE MFM and RLL Disk CD ROM Tape Floppy Support;"
   c. "Include IDE ATAPI CD ROM Support;"
   d. "Include IDE ATAPI Floppy Support;"
   e. "CMD640 Chipset Support;"
   f. "CMD640 Enhanced Support;"

g. "RZ 1000 Chipset Bug Fix Support;"

h. "Intel 82371 PUX Triton 12DMA support;"

i. "Loopback Device Support;"

j. "Multiple Devices Driver Support;"

k. "Loadable Module Linear Mode Support;

l. "RAM Disk Support;" and

m. "Initial RAM Disk INIT RD Support;"

6. In "Networking" options, "Yes" is selected for the following options:

a. "TCPIP Networking;"

b. "IP Drop Source Rocket Frames;"

c. "Network Device Support" (this option enables use of Ethernet LAN interface card);

d. "Module Dummy Network Driver Support;"

e. "EQL Serial Line Load Balancing Support as a Module;"

f. "PPP Support;"

g. "Ethernet 10/100 Mega Bit;"

h. "3-Com ISA/EISA/PCI Cards;"

i. "3C509, 3C579 Support" (this option is selected based on the type of Ethernet LAN interface card used);

j. "3C590, 3C900 Series Votrex Boomerang Support;" and

k. "PCI Ethernet Adapters."

7. In "Networking" options, "No" is selected for the following options:

a. "SCSI Support" (other SCSI support related options are automatically disabled when "SCSI Support" is not selected); and

b. "ISDN Support."

8. In "CD ROM Drivers" options, "Yes" is selected for the following options:

a. "Non-SCSI IDE ATAPI CD ROM Drives;" and

b. "Soft Configurable CD ROM Interface Card Support."

9. In "File Systems" options, "Yes" is selected for the following options:

a. "Module Minix File System as a Module;"

b. "Extended FS Support as a Module;"

c. "Second Extended File Support;" and

d. "Native Language Support Unicode Code Pages."

10. In "Character Devices" options, "Yes" is selected for the following options:

a. "Standard Generic Serial Support;"

b. "Stallion Multiple Serial Support;"

c. "Parallel Printer Support;"

d. "Module I08+ Card Support;" and

e. "PS/2 Mouse Support;"

11. In "Character Devices" options, "No" is selected for the following options:

a. "Sound Support;" and

b. "Kernel Hacking."

APPENDIX B

```
//Start of "asciiModbus.cc" Program Code

// ASCII Modbus protocol interface routines
// J.D.A. Lambert May 5, 1996
// Daniel Industries
//

#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <termios.h>
#include <sys/file.h>
#ifdef DEBUG
#include <string.h>
#endif
#ifdef TIME_DEBUG
#include <sys/time.h>
#endif
#include "asciiModbus.h"

// Local definitions
#define ONESECOND 10            // One second in 100 mS increments

// Local function prototypes

static int getInput(int fd, int buflen, char *buf,int wait);
static int sendModbusMessage(int fd,uchar addr,uchar funct,modReg *data,int nRegs);
static int readModbusMessage(int fd,modReg *data,int maxRegs, int maxTime);
static uchar hexToNibble(char hex);
```

```
#ifdef TIME_DEBUG
// Local debug data
static struct itimerval turnRoundTime;
static int first;          // Flag to show first turnround time
#endif

#ifdef REGRESSION          // A simple regression test

int main(int argc, char** argv)
{
 int fd;
 modReg rxBuf[256];
 modReg dataBuf[] = {{{(ushort)9001}},
                     {{(ushort)50}}};
 if (argc != 2) {          // Make sure device is specified
  fprintf(stderr, "Serial device name is required as a parameter\n");
  exit(1);
 }
 if ((fd = setupSerial(argv[1])) < 0) { // Open the device
  fprintf(stderr, "%s: %s\n", argv[1], sys_errlist[errno]);
  exit(1);
 }
 while (1) {
  int nchars;
  nchars = sendAndReceiveAscii(fd,1,3,
                               dataBuf,DIM(dataBuf),rxBuf,sizeof(rxBuf));
  if (nchars <= 0) {
   fprintf(stderr,"sendAndReceiveAscii timed out\n");
  }
 }
 close(fd);                // Close device if we ever get here
}
```

```
#endif

int sendAndReceiveAscii(int fd,
                        uchar addr,
                        uchar funct,
                        modReg *txData, int nRegs,
                        modReg *rxData, int maxRegs)
{
 if (flock(fd,LOCK_EX) < 0){          // This process must have exclusive use
  fprintf(stderr, "flock failed: %s\n", sys_errlist[errno]);
  exit(1);
 }
 if (sendModbusMessage(fd,addr,funct,txData,nRegs) < 0) { // Attempt to send
  fprintf(stderr, "sendModbusMessage failed: %s\n", sys_errlist[errno]);
  exit(1);
 }
#ifdef TIME_DEBUG
 turnRoundTime.it_interval.tv_sec=100;       // 100 second interval
 turnRoundTime.it_interval.tv_usec=0;
 turnRoundTime.it_value.tv_sec=100;          // and initial time
 turnRoundTime.it_value.tv_usec=0;
 setitimer(ITIMER_REAL,&turnRoundTime,NULL);
 first=1;                        // Set flag for reporting turnaround time
#endif
 int nread =
  readModbusMessage(fd,rxData,maxRegs,ONESECOND); // Try to get reply
 flock(fd,LOCK_UN);                  // Let others use this device
 usleep(10);                     // Others cannot get in without this
 return nread;                   // Tell caller what happened
}
```

```
static int getInput(int fd, int buflen, char *buf, int wait) {
 int nchars = 0;
 while ((nchars <=0) && (wait-- >= 0)) {
  nchars = read(fd,buf,buflen); // Attempt to read
  if (nchars > 0) {              // Only +ve results are meaningful
#ifdef TIME_DEBUG
   if (first) {            // Only report the first successful read
      first=0;             // Make sure others are not reported
      getitimer(ITIMER_REAL,&turnRoundTime);
      double elapsed=(100.0 -
                      ((double)turnRoundTime.it_value.tv_sec +
                       ((double)turnRoundTime.it_value.tv_usec
                        / (double)1000000.0)));
      printf("Turnaround time is %7.3f milliseconds\n",
          elapsed * 1000);
   }
#endif
#ifdef DEBUG
   printf ("%d characters read\n",nchars);
   for (int i=0;i < nchars;i++) {
      printf ("%02X,",buf[i] & 0xff);
   }
   printf("\n");
#endif
  }
  else {
   if (errno != EAGAIN) {    // Error "try again" is not really an error
      fprintf(stderr,"Fatal error during read: %s\n",
            sys_errlist[errno]);
      exit(1);          // Treat all errors as fatal for now
   }
   usleep(100000);              // Just nothing for us, wait 100 mS
 }
```

```
 }
 return nchars;
}

int setupSerial (const char *dev) {
 struct termios tios;
 int fd = open(dev,O_RDWR | O_NDELAY );
 if (fd < 0) {
  fprintf(stderr, "%s: %s\n", dev, sys_errlist[errno]);
  exit(1);
 }
 if (tcgetattr(fd, &tios) < 0) {
  fprintf(stderr,"Could not get terminal attributes: %s",sys_errlist[errno]);
  exit(1);
 }

 tios.c_cflag = CS7 |          // Seven data bits
                               // Implicitly no parity, one stop bit
  CREAD |                      // Enable Receiver
  HUPCL |                      // Hangup after close
  CLOCAL |                     // Ignore modem control lines
  PARENB;                      // Enable parity (even by default)

 tios.c_iflag    = IGNBRK | INPCK; // Ignore break check input parity errors
 tios.c_oflag    = 0;
 tios.c_lflag    = 0;
 for(int i = 0; i < NCCS; i++) {
  tios.c_cc[i] = '\0';         // no special characters
 }
 tios.c_cc[VMIN] = 1;
 tios.c_cc[VTIME] = 0;

 cfsetospeed (&tios, B9600);
```

```
 cfsetispeed (&tios, B9600);

 if (tcsetattr(fd, TCSAFLUSH, &tios) < 0) {
  fprintf(stderr,"Could not set attributes: ,%s",sys_errlist[errno]);
  exit(1);
 }
 return fd;
}

//
// Send a Modbus message contained in msg , of length len, to device fd;
//

int sendModbusMessage(int fd,uchar addr,uchar funct,modReg *msg,int nRegs) {
 uint lrc=0;                    // lrc check
 char txBuffer[512];            // Temporary transmit buffer
#ifdef DEBUG
 bzero(txBuffer,sizeof(txBuffer)); // Clear array for easy debug
#endif
 txBuffer[0] = ':';             // Package header
 sprintf(&txBuffer[1],"%02X%02X",addr,funct);
 lrc+=addr;
 lrc+=funct;
 for (int i=0;i < nRegs;i++) {  // Copy the message
  sprintf(&txBuffer[5 + (i * 4)], // to the transmit buffer
          "%02X%02X",
          msg[i].parts.msb,msg[i].parts.lsb); // Converting it to hexadecimal
  lrc+=msg[i].parts.msb;        // Account the lrc
  lrc+=msg[i].parts.lsb;
 }
 int txByteLen=5 + (nRegs * 4); // Colon(1) + addr(2) + funct(2) + (nRegs * 4)
 lrc = ~lrc +1;                 // Twos complement
 lrc &= 0xFF;                   // Mask off overflow bits
```

```
 sprintf(&txBuffer[txByteLen], // and put it into the transmitter
        "%02X\r\n",lrc); // together with the CRLF
 int nchars = write(fd,txBuffer,txByteLen + 4); // Transmit the lot
#ifdef DEBUG
 printf ("%d characters written\n",nchars);
 txBuffer[nchars]='\0';
 printf("%s",txBuffer);
 for (int i=0;i < nchars;i++) {
  printf ("%02X,",txBuffer[i] & 0xff);
 }
 printf("\n");
#endif
 return nchars;
}

//
// Read a modbus message
//
static int readModbusMessage(int fd, modReg *data,int maxRegs, int maxtime) {
 uint lrc=0;                    // Lrc accumulator
 int endPos=0;                  // Flag to mark position of end newline
 int totChars=0;                // Total characters received
 char rxBuf[512];               // Raw receive buffer
 char byteBuf[256];             // Buffer for bytes in message body
#ifdef DEBUG
 bzero(rxBuf,sizeof(rxBuf));
 bzero(byteBuf,sizeof(byteBuf));
#endif
 while ((maxtime-- > 0) &&
        (endPos == 0)) {        // Keep trying to get data every 100mS
  int nchars = getInput(fd,
                        sizeof(rxBuf) - totChars,
                        rxBuf + totChars,
```

```
                    1);
  if (nchars > 0) {          // Only deal with real data
   totChars += nchars;       // Adjust count
  }
  for (int i=0; i < totChars;i++) {
   if (rxBuf[i] == '\n') {
      endPos = i;
      break;                 // No need to go farther
   }
  }
}
if ((endPos != 0) &&         // If we have characters
  (rxBuf[0] == ':') &&       // Start colon
  (rxBuf[endPos-1] == '\r') && // and Carriage-return
  (rxBuf[endPos] == '\n')    // and Newline all in correct places
  ) {                        // We have a correctly terminated buffer
 int byteCount=0;            // Number of bytes in message
 for (int i=1;i < endPos -2;i += 2) { // Do not process ':','\r', or '\n'
  byteCount++;
  lrc += byteBuf[(i - 1)/2] = hexToNibble(rxBuf[i]) * 16 +
      hexToNibble(rxBuf[i + 1]); // Convert hex nibbles to binary bytes
 }
 if ((lrc & 0xFF) == 0) {    // If we have a good lrc
  int nRegs=0;               // Number of registers populated
  for (int i=3;i < byteCount;i += 2) { // Get modbus registers skipping
                             // address, function code, and byte count
      data->parts.msb=byteBuf[i];
      data->parts.lsb=byteBuf[i + 1];
      data++;                // Next register
      if (nRegs++ >= maxRegs) { // Tally registers and if too many..
       break;                // quit
      }
  }
```

```
      if (--nRegs <= maxRegs) {
          return nRegs;          // It's a good message
      }
     }
    }
    return -1;                   // A bummer by default
}

//
// Convert a hex character into an unsigned character nibble
//
static unsigned char hexToNibble(char hex)
{
 if ((hex >= 'A') && (hex <= 'F')) {
  return hex - 'A' + 10;
 }
 else if ((hex >= '0') && (hex <= '9')) {
  return hex - '0';
 }
 else {
  fprintf(stderr,"Bad hex nibble %c\n",hex);
  exit(1);
 }
}
// End of "asciiModbus.cc" Program Code


// Start of "convertModbus.cc" Program Code
//
//
// File: convertModbus.cc
// Author: David Lambert
// Date: June 30, 1988
```

```
//
// Various functions to convert modbus registers to and from native types.
//

#include "convertModbus.h"

// Put a Short s to a modbus register m

void putShort(short s,modReg *m) {
 m->value=s;
}

// Get a short from a modbus stream m

short getShort(modReg *m)
{
 return m->value;
}

// Put a float f to a modbus stream m

void putFloat(float f,modReg *m)
{
 union {
  struct {modReg hi,lo;}  c;   // components
  float f;
 } u;
 u.f=f;                    // Load up the union
 m[0].value=u.c.lo.value;      // Sign and exponent
 m[1].value=u.c.hi.value;      // mantissa
}

// Return a float from the modbus stream m
```

```
float getFloat(modReg *m)
{
 union {
  struct {modReg hi,lo;}  c; // components
  float f;
 } u;
 u.c.lo.value=m[0].value;       // Sign and exponent
 u.c.hi.value=m[1].value;       // Rest of mantissa
 return (u.f);
}

// End of "convertModbus.cc" Program Code


// Start of "asciiModbus.h" Program Code

/*
 File: asciiModbus.h
*/

#ifndef ASCIIMODBUS_H
#define ASCIIMODBUS_H

#define DIM(x) (sizeof(x)/sizeof(x[0]))

typedef unsigned int uint;
typedef unsigned short ushort;
typedef unsigned char uchar;

typedef union {                         // Modbus register union
 ushort value;                  // The value as seen by this machine
 struct {
```

```
    uchar lsb;
    uchar msb;
   } parts;                    // Individual parts
  } modReg;                    // Modbus register

int setupSerial (const char *dev); // Opens and sets up device

int sendAndReceiveAscii(int fd, // File descriptor of serial channel
                        uchar addr, // Modbus address
                        uchar funct, // Modbus function code
                        modReg *tx, // Modbus register array to transmit
                        int nRegs, // Number of registers in tx array
                        modReg *rx, // Repository for received registers
                        int maxRegs); // Maximum number of rx registers

#endif

// End of "asciiModbus.h" Program Code

// Start of "convertModbus.h" Program Code

#ifndef CONVERTMODBUS_H
#define CONVERTMODBUS_H
//
//
// File: convertModbus.h
// Author: David Lambert
// Date: June 30, 1988
//
```

```
// Various functions to convert modbus registers to and from native types.
//

#include "asciiModbus.h"

// Put a Short s to a modbus register m

void putShort(short s,modReg *m);

// Get a short from a modbus stream m

short getShort(modReg *m);

// Put a float f to a modbus stream m

void putFloat(float f,modReg *m);


// Return a float from the modbus stream m

float getFloat(modReg *m);

#endif

// End of "convertModbus.h" Program Code
```

APPENDIX C

```
// Start of "Makefile.2" Program Code

CC=g++

all: flowLog

flowLog.o: flowLog.cc ../asciiModbus.h ../modData.h Makefile
        $(CC) -c -DDEBUG -Wall -g flowLog.cc

flowLog: ../asciiModbus.o ../convertModbus.o flowLog.o Makefile
        $(CC) ../asciiModbus.o ../convertModbus.o flowLog.o -o flowLog

// End of "Makefile.2" Program Code


// Start of "flowLog.cc" Program Code

// Ultrasonic flow log acquisition module
// D. Lambert
// Daniel Industries
// June 8, 1998


#include <stdio.h>
#include <errno.h>
#include <unistd.h>
#include <time.h>
#include "../modData.h"
#include "../convertModbus.h"
```

```
#include "../asciiModbus.h"

// Static function prototypes


// Local definitions

#define READREGS 3              // Read multiple registers function code
#define NTXREGS 2               // Number of registers to transmit


int main(int argc, char **argv)
{
 int logRate=60;                // Default to one minute logging rate
 int fd;
 int address;
 modReg rxBuf[256];
 modReg txData[2];              // Transmit data buffer
 if (argc < 2) {        // Make sure device is specified
  fprintf(stderr, "Serial device name is required as parameter 1\n");
  exit(1);
 }
 if (argc < 3) {
  fprintf(stderr, "Modbus ID for meter is required as parameter 2\n");
  exit(1);
 }
 sscanf(argv[2],"%ud",&address);
 if (address > 64) {
  fprintf(stderr, "Address %d is out of range\n",address);
  exit(1);
 }
 if ((fd = setupSerial(argv[1])) < 0) { // Open the device
  fprintf(stderr, "Failed to open %s: %s\n", argv[1], sys_errlist[errno]);
```

```
 exit(1);
}
if (argc < 4) {
 fprintf(stderr, "Log file name is required as parameter 3");
 exit(1);
}
if (argc > 4) {          // We have a logging rate
 sscanf(argv[4],"%d",&logRate);
}
FILE *logFile = NULL;              // Logging file ID
txData[0].value = 392;        // Start Register
txData[1].value = 2;          // Number of registers
while(TRUE) {                      // Forever
 int nRegsRx = sendAndReceiveAscii(fd,address,
                                   READREGS,
                                   txData,NTXREGS,
                                   rxBuf,sizeof(rxBuf));
if (nRegsRx <= 0) {
 fprintf(stderr,"Communication Failure:- %s\n",sys_errlist[errno]);
}
else {                         // We have good data, process it!
 static int lastHour = -1;
 time_t timeNow=time(NULL);
 struct tm *tmNow=localtime(&timeNow);
 if ((tmNow->tm_hour == 0) && (lastHour != 0)) { // Day change
     remove(argv[3]);       // For now just loose previous data
     logFile=NULL;
 }
 lastHour = tmNow->tm_hour; // Remember hour
 if (logFile == NULL) {     // We need to try and open the log file
     if ((logFile = fopen(argv[3],"a+")) == NULL) {
      fprintf(stderr, "Failed to open log file: %s\n",sys_errlist[errno]);
      exit(1);
```

```
        }
      }
      char dateTimeString[256]; // Date and time string
      strftime(dateTimeString,sizeof(dateTimeString),"%H:%M:%S %m/%d/%y",
                  tmNow);
      modReg* mrp=rxBuf;      // Pointer to rx modbus registers (only 1!)
      fprintf(logFile,"%s, %f\n", // Print date and flow rate
            dateTimeString,getFloat(mrp));
      fclose(logFile);          // Close it and...
      logFile=NULL;             // try again later
     }
     sleep(logRate);            // Wait a minute till next log
    }
    fflush(NULL);                   // Flush all streams
    close(fd);                // Close device
}

// End of "flowLog.cc" Program Code
```

APPENDIX D

```
// Start of "ultrasonic.cc" Program Code


// Ultrasonic acquisition module
// D. Lambert
// Daniel Industries
// June 8, 1998


#include <stdio.h>
#include <errno.h>
#include <unistd.h>
#include "modData.h"
#include "convertModbus.h"
#include "asciiModbus.h"
#include "ultrasonicMessageBlocks.h"

// Static function prototypes


// Local definitions

#define READREGS 3              // Read multiple registers function code


int main(int argc, char **argv)
{
 int fd;
 uint mbno;                     // Message block number
 int address;
 messageBlock *mb;
```

```
modReg rxBuf[256];
modReg txData[2];              // Transmit data buffer
if (argc < 2) {        // Make sure device is specified
 fprintf(stderr, "Serial device name is required as parameter 1\n");
}
if (argc < 3) {
 fprintf(stderr, "Modbus ID for meter is required as parameter 2\n");
}
if (argc < 4) {
 fprintf(stderr, "Message block number is required as parameter 3\n");
 exit(1);
}
sscanf(argv[2],"%ud",&address);
if (address > 64) {
 fprintf(stderr, "Address %d is out of range\n",address);
 exit(1);
}
sscanf(argv[3],"%ud",&mbno);
if (mbno >= nMessageBlocks) {
 fprintf(stderr, "Message block number %d is out of range\n",mbno);
 exit(1);
}
if ((mb = messageBlocks[mbno]) == NULL) {
 fprintf(stderr, "Message block number %d is not yet implemented\n",mbno);
 exit(1);
}
if ((fd = setupSerial(argv[1])) < 0) {// Open the device
 fprintf(stderr, "%s: %s\n", argv[1], sys_errlist[errno]);
 exit(1);
}
txData[0].value = mb->points[0].reg; // Start Register
txData[1].value = registersInMessageBlock(mb); // Number of registers
int nRegsRx = sendAndReceiveAscii(fd,address,
```

```
                                READREGS,
                                txData,2,
                                rxBuf,sizeof(rxBuf));
    if (nRegsRx <= 0) {
     fprintf(stderr,"Communication Failure:- %s\n",sys_errlist[errno]);
     printf("Communication Failure:- %s\n",sys_errlist[errno]);
    }
    else {                  // We have good data, process it!
     modReg* mrp=rxBuf;     // Pointer to rx modbus registers
     for (int i=0;i < nRegsRx;i++) {
      modData* mdp=&mb->points[i]; // Pointer to modbus data descriptors
      if (mdp->type == 'F') {     // Process floats
          if (mdp->display) {
           printf("<tr><td>%8s<td>%f<td>%s</tr>\n",mdp->name,getFloat(mrp),mdp->units);
          }
          mrp += 2;             // A float is two modbus registers
      }
      else if (mdp->type == 'I') { // Process ints
          if (mdp->display) {
           printf("<tr><td>%8s<td>%hd<td>%s</tr>\n",mdp->name,getShort(mrp),mdp->units);
          }
          mrp++;                // A short integer is one modbus register
      }
     }
    }
    fflush(NULL);               // Flush all streams
    close(fd);                  // Close device
}


// End of "ultrasonic.cc" Program Code
```

```
// Start of "ultrasonicMessageBlocks.cc" Program Code


//
// File: ultrasonicMessageBlocks.cc
// Author D. Lambert
// Date: July 2, 1998
//
// Defines message blocks as
// implemented in the ultrasonic meter.
//

#include <unistd.h>
#include <stdio.h>
#include "ultrasonicMessageBlocks.h"

// Message Block Definitions

static messageBlock messageBlock1 = {       // Message Block Definitions
 {
  {1,"Spare","",'I',NO},
  {2,"Parity","",'I',YES},
  {3,"BaudRate","Bits/sec",'I',YES},
  {4,"WdLength","Bits/word",'I',YES},
  {5,"StopBits","Bits",'I',YES},
  {6,"ModbusID","",'I',YES},
  {7,"Spare","",'I',NO},
  {8,"Spare","",'I',NO},
  {9,"Spare","",'I',NO},
  {10,"Spare","",'I',NO},
  {11,"Spare","",'I',NO},
  {12,"Spare","",'I',NO},
  {13,"Spare","",'I',NO},
```

```
    {14,"Spare","",I',NO},
    {15,"Spare","",I',NO}
  },
  15                              // Number of points
};

static messageBlock messageBlock9 = { // Oper
  {
    {200,"ZeroCut","ms",'F',YES},
    {202,"SSMax","ms",'F',YES},
    {204,"SSMin","ms",'F',YES},
    {206,"EmRate","sec",'F',YES},
    {208,"SampRate","MHz",'F',YES},
    {210,"PulseWidth","sec",'F',YES},
    {212,"PctFail","%",'F',YES},
    {214,"MinHoldTime","sec",'F',YES},
    {216,"Pk1Pct","%",'F',YES},
    {218,"MinSigQlty","",'F',YES},
    {220,"DltChk","sec",'F',YES},
    {222,"Spare","",'F',NO},
    {224,"NegSpan","sec",'F',YES},
    {226,"PosSpan","sec",'F',YES},
    {228,"TmDevLow1","sec",'F',YES},
    {230,"TmDevLow2","sec",'F',YES},
    {232,"CRange","%",'F',YES},
    {234,"SDevLow","sec",'F',YES},
    {236,"SDevFctr","",'F',YES},
    {238,"Pk1Wdth","sec",'F',YES},
    {240,"TmDevFctr1","",'F',YES},
    {242,"TmDevFctr2","",'F',YES},
    {244,"FlwChgThrsh","%",'F',YES},
    {246,"FlwChgDtctr","sec",'F',YES},
    {248,"StackEmRate","sec",'F',YES}},
```

```
    25                          // Number of points
};

static messageBlock messageBlock12 = {
  {
    {350,"Spare","",'F',NO},
    {352,"FlowVelA","m/s",'F',YES},
    {354,"FlowVelB","m/s",'F',YES},
    {356,"FlowVelC","m/s",'F',YES},
    {358,"FlowVelD","m/s",'F',YES},
    {360,"AvgFlow","m/s",'F',YES},
    {362,"SndVelA","m/s",'F',YES},
    {364,"SndVelB","m/s",'F',YES},
    {366,"SndVelC","m/s",'F',YES},
    {368,"SndVelD","m/s",'F',YES},
    {370,"AvgSndVel","m/s",'F',YES},
    {372,"CalVol","m^3",'F',NO},
    {374,"CalTime","pulses",'F',NO},
    {376,"SDevVelA","m/s",'F',NO},
    {378,"SDevVelB","m/s",'F',NO},
    {380,"SDevVelC","m/s",'F',NO},
    {382,"SDevVelD","m/s",'F',NO},
    {384,"SDevSndA","m/s",'F',NO},
    {386,"SDevSndB","m/s",'F',NO},
    {388,"SDevSndC","m/s",'F',NO},
    {390,"SDevSndD","m/s",'F',NO},
    {392,"FlowRate","acmh",'F',YES},
    {394,"CutRate","acmh",'F',NO},
    {396,"Cal VolA","m^3",'F',NO},
    {398,"Cal Volb","m^3",'F',NO}},
    25                          // Number of points
};
```

```
static messageBlock messageBlock13 = { // Oper
 {
  {400,"Spare","---",'F',YES},
  {402,"MaxTmA1","sec",'F',YES},
  {404,"MaxTmB1","sec",'F',YES},
  {406,"MaxTmC1","sec",'F',YES},
  {408,"MaxTmD1","sec",'F',YES},
  {410,"MaxTmA2","sec",'F',YES},
  {412,"MaxTmB2","sec",'F',YES},
  {414,"MaxTmC2","sec",'F',YES},
  {416,"MaxTmD2","sec",'F',YES},
  {418,"MinTmA1","sec",'F',YES},
  {420,"MinTmB1","sec",'F',YES},
  {422,"MinTmC1","sec",'F',YES},
  {424,"MinTmD1","sec",'F',YES},
  {426,"MinTmA2","sec",'F',YES},
  {428,"MinTmB2","sec",'F',YES},
  {430,"MinTmC2","sec",'F',YES},
  {432,"MinTmD2","sec",'F',YES},
  {434,"MeanTmA1","sec",'F',YES},
  {436,"MeanTmB1","sec",'F',YES},
  {438,"MeanTmC1","sec",'F',YES},
  {440,"MeanTmD1","sec",'F',YES},
  {442,"MeanTmA2","sec",'F',YES},
  {444,"MeanTmB2","sec",'F',YES},
  {446,"MeanTmC2","sec",'F',YES},
  {448,"MeanTmD2","sec",'F',YES}},
 25                          // Number of points
};


static messageBlock messageBlock14 = { // Calculation Results
 {
```

```
    {450,"Spare","---",'F',NO},
    {452,"DltTmA","sec",'F',YES},
    {454,"DltTmB","sec",'F',YES},
    {456,"DltTmC","sec",'F',YES},
    {458,"DltTmD","sec",'F',YES},
    {460,"SDevTmA1","ns",'F',YES},
    {462,"SDevTmB1","ns",'F',YES},
    {464,"SDevTmB1","ns",'F',YES},
    {466,"SDevTmD1","ns",'F',YES},
    {468,"SDevTmA2","ns",'F',YES},
    {470,"SDevTmB2","ns",'F',YES},
    {472,"SDevTmC2","ns",'F',YES},
    {474,"SDevTmD2","ns",'F',YES},
    {476,"SDevDltTmA","ns",'F',YES},
    {478,"SDevDltTmB","ns",'F',YES},
    {480,"SDevDltTmC","ns",'F',YES},
    {482,"SDevDltTmD","ns",'F',YES},
    {484,"MaxDltTmA","sec",'F',YES},
    {486,"MaxDltTmB","sec",'F',YES},
    {488,"MaxDltTmC","sec",'F',YES},
    {490,"MaxDltTmD","sec",'F',YES},
    {492,"MinDltTmA","sec",'F',YES},
    {494,"MinDltTmB","sec",'F',YES},
    {496,"MinDltTmC","sec",'F',YES},
    {498,"MinDltTmD","sec",'F',YES}},
   25                              // Number of points
};


static messageBlock messageBlock15 = { // Diag
  {
   {500, "Spare", "---", 'F', NO},
   {502, "HoldTmA1", "sec", 'F', YES},
```

```
    {504, "HoldTmA2", "sec", 'F', YES},
    {506, "HoldTmB1", "sec", 'F', YES},
    {508, "HoldTmB2", "sec", 'F', YES},
    {510, "HoldTmC1", "sec", 'F', YES},
    {512, "HoldTmC2", "sec", 'F', YES},
    {514, "HoldTmD1", "sec", 'F', YES},
    {516, "HoldTmD2", "sec", 'F', YES},
    {518, "WdwPtrA1", "si", 'F', YES},
    {520, "WdwPtrA2", "si", 'F', YES},
    {522, "WdwPtrB1", "si", 'F', YES},
    {524, "WdwPtrB2", "si", 'F', YES},
    {526, "WdwPtrC1", "si", 'F', YES},
    {528, "WdwPtrC2", "si", 'F', YES},
    {530, "WdwPtrD1", "si", 'F', YES},
    {532, "WdwPtrD2", "si", 'F', YES},
    {534, "Spare", "---", 'F', NO},
    {536, "Spare", "---", 'F', NO},
    {538, "Spare", "---", 'F', NO},
    {540, "Spare", "---", 'F', NO},
    {542, "Spare", "---", 'F', NO},
    {544, "Spare", "---", 'F', NO},
    {546, "Spare", "---", 'F', NO},
    {548, "Spare", "---", 'F', NO}},
  25                              // Number of points
};


static messageBlock messageBlock16 = { // Diag
  {
    {550, "PfA1", "si", 'F', YES},
    {552, "PfA2", "si", 'F', YES},
    {554, "PfB1", "si", 'F', YES},
    {556, "PfB2", "si", 'F', YES},
```

```
    {558, "PfC1", "si", 'F', YES},
    {560, "PfC2", "si", 'F', YES},
    {562, "PfD1", "si", 'F', YES},
    {564, "PfD2", "si", 'F', YES},
    {566, "P1A1", "si", 'F', YES},
    {568, "P1A2", "si", 'F', YES},
    {570, "P1B1", "si", 'F', YES},
    {572, "P1B2", "si", 'F', YES},
    {574, "P1C1", "si", 'F', YES},
    {576, "P1C2", "si", 'F', YES},
    {578, "P1D1", "si", 'F', YES},
    {580, "P1D2", "si", 'F', YES},
    {582, "PwA1", "si", 'F', YES},
    {584, "PwA2", "si", 'F', YES},
    {586, "PwB1", "si", 'F', YES},
    {588, "PwB2", "si", 'F', YES},
    {590, "PwC1", "si", 'F', YES},
    {592, "PwC2", "si", 'F', YES},
    {594, "PwD1", "si", 'F', YES},
    {596, "PwD2", "si", 'F', YES},
    {598, "Spare", "---", 'F', YES}},
    25                          // Number of points
};

static messageBlock messageBlock17 = { // Calculation Results
    {
    {600, "PropA", "---", 'F', YES},
    {602, "PropB", "---", 'F', YES},
    {604, "PropC", "---", 'F', YES},
    {606, "PropD", "---", 'F', YES},
    {608, "RPropA", "---", 'F', YES},
    {610, "RPropB", "---", 'F', YES},
    {612, "RPropC", "---", 'F', YES},
```

```
    {614, "RPropD", "---", 'F', YES},
    {616, "RABLo", "---", 'F', YES},
    {618, "RABHi", "---", 'F', YES},
    {620, "RDCLo", "---", 'F', YES},
    {622, "RDCHi", "---", 'F', YES},
    {624, "RADLo", "---", 'F', YES},
    {626, "RADHi", "---", 'F', YES},
    {628, "RBCLo", "---", 'F', YES},
    {630, "RBCHi", "---", 'F', YES},
    {632, "Spare", "---", 'F', NO},
    {634, "Spare", "---", 'F', NO},
    {636, "Spare", "---", 'F', NO},
    {638, "Spare", "---", 'F', NO},
    {640, "Spare", "---", 'F', NO},
    {642, "Spare", "---", 'F', NO},
    {644, "Spare", "---", 'F', NO},
    {646, "Spare", "---", 'F', NO},
    {648, "Spare", "---", 'F', NO}},
  25                              // Number of points
};

static messageBlock messageBlock18 = { // Oper
  {
    {650, "Spare", "---", 'F', NO},
    {652, "Spare", "---", 'F', NO},
    {654, "Spare", "---", 'F', NO},
    {656, "Spare", "---", 'F', NO},
    {658, "Spare", "---", 'F', NO},
    {660, "Spare", "---", 'F', NO},
    {662, "Spare", "---", 'F', NO},
    {664, "Spare", "---", 'F', NO},
    {666, "Spare", "---", 'F', NO},
```

```
    {668, "Spare", "---", 'F', NO},
    {670, "Spare", "---", 'F', NO},
    {672, "Spare", "---", 'F', NO},
    {674, "Spare", "---", 'F', NO},
    {676, "Spare", "---", 'F', NO},
    {678, "Spare", "---", 'F', NO},
    {680, "Spare", "---", 'F', NO},
    {682, "Spare", "---", 'F', NO},
    {684, "Spare", "---", 'F', NO},
    {686, "Spare", "---", 'F', NO},
    {688, "Spare", "---", 'F', NO},
    {690, "Spare", "---", 'F', NO},
    {692, "Spare", "---", 'F', NO},
    {694, "Spare", "---", 'F', NO},
    {696, "Spare", "---", 'F', NO},
    {698, "Spare", "---", 'F', NO}},
  25                              // Number of points
};


static messageBlock messageBlock19 = { // Diag
  {
    {700, "Wdw1A1", "---", 'F', YES},
    {702, "Wdw1A2", "---", 'F', YES},
    {704, "Wdw1B1", "---", 'F', YES},
    {706, "Wdw1B2", "---", 'F', YES},
    {708, "Wdw1C1", "---", 'F', YES},
    {710, "Wdw1C2", "---", 'F', YES},
    {712, "Wdw1D1", "---", 'F', YES},
    {714, "Wdw1D2", "---", 'F', YES},
    {716, "Wdw2A1", "---", 'F', YES},
    {718, "Wdw2A2", "---", 'F', YES},
    {720, "Wdw2B1", "---", 'F', YES},
```

```
    {722, "Wdw2B2", "---", 'F', YES},
    {724, "Wdw2C1", "---", 'F', YES},
    {726, "Wdw2C2", "---", 'F', YES},
    {728, "Wdw2D1", "---", 'F', YES},
    {730, "Wdw2D2", "---", 'F', YES},
    {732, "Spare", "---", 'F', NO},
    {734, "Spare", "---", 'F', NO},
    {736, "Spare", "---", 'F', NO},
    {738, "Spare", "---", 'F', NO},
    {740, "Spare", "---", 'F', NO},
    {742, "Spare", "---", 'F', NO},
    {744, "Spare", "---", 'F', NO},
    {746, "Spare", "---", 'F', NO},
    {748, "Spare", "---", 'F', NO}},
  25                          // Number of points
};


static messageBlock messageBlock20 = { // Diag
  {
    {750, "Wdw3A1", "---", 'F', YES},
    {752, "Wdw3A2", "---", 'F', YES},
    {754, "Wdw3B1", "---", 'F', YES},
    {756, "Wdw3B2", "---", 'F', YES},
    {758, "Wdw3C1", "---", 'F', YES},
    {760, "Wdw3C2", "---", 'F', YES},
    {762, "Wdw3D1", "---", 'F', YES},
    {764, "Wdw3D2", "---", 'F', YES},
    {766, "Wdw4A1", "---", 'F', YES},
    {768, "Wdw4A2", "---", 'F', YES},
    {770, "Wdw4B1", "---", 'F', YES},
    {772, "Wdw4B2", "---", 'F', YES},
    {774, "Wdw4C1", "---", 'F', YES},
```

```
  {776, "Wdw4C2", "---", 'F', YES},
  {778, "Wdw4D1", "---", 'F', YES},
  {780, "Wdw4D2", "---", 'F', YES},
  {782, "Spare", "---", 'F', NO},
  {784, "Spare", "---", 'F', NO},
  {786, "Spare", "---", 'F', NO},
  {788, "Spare", "---", 'F', NO},
  {790, "Spare", "---", 'F', NO},
  {792, "Spare", "---", 'F', NO},
  {794, "Spare", "---", 'F', NO},
  {796, "Spare", "---", 'F', NO},
  {798, "Spare", "---", 'F', NO}},
 25                              // Number of points
};

static messageBlock messageBlock21 = { // Diag
 {
  {800, "SpecMaxA1", "---", 'F', YES},
  {802, "SpecMaxA2", "---", 'F', YES},
  {804, "SpecMaxB1", "---", 'F', YES},
  {806, "SpecMaxB2", "---", 'F', YES},
  {808, "SpecMaxC1", "---", 'F', YES},
  {810, "SpecMaxC2", "---", 'F', YES},
  {812, "SpecMaxD1", "---", 'F', YES},
  {814, "SpecMaxD2", "---", 'F', YES},
  {816, "MaxNEA1", "---", 'F', YES},
  {818, "MaxNEA2", "---", 'F', YES},
  {820, "MaxNEB1", "---", 'F', YES},
  {822, "MaxNEB2", "---", 'F', YES},
  {824, "MaxNEC1", "---", 'F', YES},
  {826, "MaxNEC2", "---", 'F', YES},
  {828, "MaxNED1", "---", 'F', YES},
```

```
    {830, "MaxNED2", "---", 'F', YES},
    {832, "QpefA1", "---", 'F', YES},
    {834, "QpefA2", "---", 'F', YES},
    {836, "QpefB1", "---", 'F', YES},
    {838, "QpefB2", "---", 'F', YES},
    {840, "QpefC1", "---", 'F', YES},
    {842, "QpefC2", "---", 'F', YES},
    {844, "QpefD1", "---", 'F', YES},
    {846, "QpefD2", "---", 'F', YES},
    {848, "Sparc", "---", 'F', NO}},
  25                          // Number of points
};


messageBlock *messageBlocks[] = {
  NULL,                       // Message block 0 is not defined
  &messageBlock1,             // Message block 1
  NULL,                       // Message block 2 is not defined
  NULL,                       // Message block 3 is not defined
  NULL,                       // Message block 4 is not defined
  NULL,                       // Message block 5 is not defined
  NULL,                       // Message block 6 is not defined
  NULL,                       // Message block 7 is not defined
  NULL,                       // Message block 8 is not defined
  &messageBlock9,             // Message block 9
  NULL,                       // Message block 10 is not defined
  NULL,                       // Message block 11 is not defined
  &messageBlock12,            // Message block 12
  &messageBlock13,            // Message block 13
  &messageBlock14,            // Message block 14
  &messageBlock15,            // Message block 15
  &messageBlock16,            // Message block 16
```

```
    &messageBlock17,        // Message block 17
    &messageBlock18,        // Message block 18
    &messageBlock19,        // Message block 19
    &messageBlock20,        // Message block 20
    &messageBlock21,        // Message block 21
    NULL,                   // Message block 22 is not defined
    NULL,                   // Message block 23 is not defined
    NULL,                   // Message block 24 is not defined
    NULL,                   // Message block 25 is not defined
    NULL,                   // Message block 26 is not defined
    NULL,                   // Message block 27 is not defined
    NULL,                   // Message block 28 is not defined
    NULL,                   // Message block 29 is not defined
    NULL,                   // Message block 30 is not defined
    NULL,                   // Message block 31 is not defined
    NULL                    // Message block 32 is not defined
};

unsigned int nMessageBlocks = (sizeof(messageBlocks) /
                    sizeof(messageBlocks[0]));

//  Method to count the number of native modbus registers in a message block

int registersInMessageBlock(messageBlock *mbp)
{
 int i;                 // Loop control
 int c=0;               // Accumulator
 for (i = 0;i < mbp->nPoints;i++) {
  switch (mbp->points[i].type) {
  case 'F': {           // Modbus Float is 2 registers
   c += 2;
   break;
  }
```

```
        case 'I': {
          c++;                          // Modbus Int is 1 register
          break;
        }
        default:
          fprintf(stderr,"Unrecognized data item type %c\n",
                  mbp->points[i].type);
        }
      }
      return c;
    }


    // End of "ultrasonicMessageBlocks.cc" Program Code


    // Start of "Makefile" Program Code


    CC=g++

    all: ultrasonic

    convertModbus.o: convertModbus.cc convertModbus.h Makefile
            $(CC) -c -Wall -g convertModbus.cc

    asciiModbus.o: asciiModbus.cc asciiModbus.h Makefile
            $(CC) -c -Wall -g asciiModbus.cc

    ultrasonicMessageBlocks.o: ultrasonicMessageBlocks.cc\
            ultrasonicMessageBlocks.h Makefile
            $(CC) -c -Wall -g ultrasonicMessageBlocks.cc
```

```
ultrasonic.o: ultrasonic.cc asciiModbus.h modData.h Makefile
        $(CC) -c -DDEBUG -Wall -g ultrasonic.cc

ultrasonic: asciiModbus.o convertModbus.o ultrasonic.o\
        ultrasonicMessageBlocks.o Makefile
        $(CC) asciiModbus.o ultrasonic.o ultrasonicMessageBlocks.o\
        convertModbus.o -o ultrasonic


// End of "Makefile" Program Code


// Start of "ultrasonicMessageBlocks.h" Program Code


#ifndef ULTRASONICMESSAGEBLOCS_H
#define ULTRASONICMESSAGEBLOCS_H

//
// File: ultrasonicMessageBlocks.h
// Author D. Lambert
// Date: July 2, 1998
//
// Defines structures for message blocks as
// implemented in the ultrasonic meter.
//
#include "modData.h"

#define MAXPOINTS 128          // Maximum number of points in a message block

typedef struct {
 modData points[MAXPOINTS];
 int nPoints;
```

```
} messageBlock;

//
// The array of message blocks
//

extern messageBlock *messageBlocks[];
extern unsigned int nMessageBlocks;

//
// Return a count of the number of modbus registers in a message block
//
int registersInMessageBlock(messageBlock *mbp);


#endif


// End of "ultrasonicMessageBlocks.h" Program Code


// Start of "ultrasonic12" Program Code


#!/bin/bash

cat << EOM
Content-type: text/html

<html>
<head>
<TITLE>Ultrasonic Meter 1</TITLE>
```

```
</head>
<BODY background="../back11a.gif">

<body>
<H1><img src="../logo_bln.gif" align=bottom></H1>
<H1>Ultrasonic Meter 1</H1>
<H1><img src="../danlineb.gif" align=bottom></H1>
<HR>
<BR>
<H2>
<CENTER>
<applet code=javachart.applet.dateLineApp.class codebase=../ width=900 height=240>
<param name=CopyrightNotification value="JavaChart is a copyrighted work, and subject to full legal protection">
    <param name=titleString value="Today's Flow History">
    <param name=titleFont value="TimesRoman,20,1">
    <param name=plotAreaTop value=".80">
    <param name=plotAreaLeft value=".05">
    <param name=plotAreaRight value=".85">
    <param name=plotAreaBottom value=".10">

    <param name=xAxisOptions value="gridOn">

    <param name=yAxisOptions value="gridOn, rightAxis">
    <param name=networkInterval value="20">

    <param name=customDatasetHandler value="../ultsonic/ultrasonic.log">
    <param name=dataset0name value="Meter1">
    <param name=dataset0Color value="red">
    <param name=legendOn value="yes">
    <param name=legendLabelFont value="TimesRoman,24,2">
    <param name=2D value="yes">
```

```
</applet>
<HR>
<table border=1 bgcolor=whitesmoke cellspacing=0 cellpadding=5 cols=3 align=left>
<tr><th colspan=3><B>Assorted Datasets</B>
<tr>
EOM
cat <<EOF
<td valign=top><table bgcolor=white border=1 cellspacing=0 cellpadding=5 cols=3>
<tr><th colspan=3><B>Setup</B></th>
EOF
/bin/ultrasonic /dev/cua0 32 1
cat <<EOF
</table>
EOF

cat <<EOF
<td valign=top><table bgcolor=white border=1 cellspacing=0 cellpadding=5 cols=3>
<tr><th colspan=3><B>Flow Data</B></th>
EOF
/bin/ultrasonic /dev/cua0 32 12
cat <<EOF
</table>
EOF

cat <<EOF
<td val=top><table bgcolor=white border=1 cellspacing=0 cellpadding=5 cols=3>
<tr><th colspan=3><B>Calculation Results</B></th>
EOF
/bin/ultrasonic /dev/cua0 32 14
cat <<EOF
</table>
EOF
```

```
cat << EOM
</table>
</H2>
<H2>Click meter Image to Refresh</H2>
<A HREF=ultrasonic12><img src="../ultsonic.gif" align=top></A>
</CENTER>
</body>
</html>
EOM


// End of "ultrasonic12" Program Code


// Start of "ultrasonic.c" Program Code


// Ultrasonic acquisition module
// D. Lambert
// Daniel Industries
// June 8, 1998


#include <stdio.h>
#include <errno.h>
#include <unistd.h>
#include "modData.h"
#include "convertModbus.h"
#include "asciiModbus.h"
#include "ultrasonicMessageBlocks.h"

// Static function prototypes
```

```
// Local definitions

#define READREGS 3              // Read multiple registers function code


int main(int argc, char **argv)
{
 int nRegsRx;                   // Number of registers received
 int fd;
 uint mbno;                     // Message block number
 int address;
 messageBlock *mb;
 modReg rxBuf[256];
 modReg txData[2];              // Transmit data buffer
 if (argc < 2) {        // Make sure device is specified
  fprintf(stderr, "Serial device name is required as parameter 1\n");
 }
 if (argc < 3) {
  fprintf(stderr, "Modbus ID for meter is required as parameter 2\n");
 }
 if (argc < 4) {
  fprintf(stderr, "Message block number is required as parameter 3\n");
  exit(1);
 }
 sscanf(argv[2],"%ud",&address);
 if (address > 64) {
  fprintf(stderr, "Address %d is out of range\n",address);
  exit(1);
 }
 sscanf(argv[3],"%ud",&mbno);
 if (mbno >= nMessageBlocks) {
  fprintf(stderr, "Message block number %d is out of range\n",mbno);
```

```
 exit(1);
}
if ((mb = messageBlocks[mbno]) == NULL) {
 fprintf(stderr, "Message block number %d is not yet implemented\n",mbno);
 exit(1);
}
if ((fd = setupSerial(argv[1])) < 0) { // Open the device
 fprintf(stderr, "%s: %s\n", argv[1], sys_errlist[errno]);
 exit(1);
}
txData[0].value = mb->points[0].reg; // Start Register
txData[1].value = registersInMessageBlock(mb); // Number of registers
nRegsRx = sendAndReceiveAscii(fd,address,
                              READREGS,
                              txData,2,
                              rxBuf,sizeof(rxBuf));
if (nRegsRx <= 0) {
 fprintf(stderr,"Communication Failure:- %s\n",sys_errlist[errno]);
 printf("Communication Failure:- %s\n",sys_errlist[errno]);
}
else {                  // We have good data, process it!
 int i;                 // Loop counter
 modReg* mrp=rxBuf;     // Pointer to rx modbus registers
 for (i=0;i < nRegsRx;i++) {
  modData* mdp=&mb->points[i]; // Pointer to modbus data descriptors
  if (mdp->type == 'F') {     // Process floats
      if (mdp->display) {
       printf("%8s\t%f %s\n",mdp->name,getFloat(mrp),mdp->units);
      }
      mrp += 2;               // A float is two modbus registers
  }
  else if (mdp->type == 'I') { // Process ints
      if (mdp->display) {
```

```
            printf("%8s\t%hd %s\n",mdp->name,getShort(mrp),mdp->units);
          }
          mrp++;                    // A short integer is one modbus register
        }
      }
    }
    fflush(NULL);                   // Flush all streams
    close(fd);                      // Close device
  }

// End of "ultrasonic.c" Program Code
```

APPENDIX E

makeLILObootdisk script

```
# This is make LILObootdisk script
# First make a compressed image of the root file system using a ram disk image
./ make CompressedRamDisk
# Unmount the flash disk
unmount /mnt/flash
# Set a variable to be larger than the number of blocks
# occupied by the kernel image and a minimal file system.
Export KERNEL_BLOCKS=430
# Make a file system that size on the first part of the FLASH disk.
Mke2fs -i 8192 -m 0 /dev/hdc $KERNEL_BLOCKS
# Now mount that file system that has just been created
mount /dev/hdc /mnt/flash
# Remove files that are not required
rm -rf /mnt/flash/lost+found
# Make the necessary directories
mkdir /mnt/flash/lost+found
# Make the necessary directories
mkdir /mnt/flash/{boot, dev}
# Copy needed devices,
cp -dpR /dev/{null, hda, hda1, hdc, hdc1} /mnt/flash/dev
# boot files
cp /boot/boot.b /mnt/flash/boot
# configuration files, and finally the kernel image
cp bdlilo.conf vmlinuz.embedded /mnt/flash
# Run lilo to update the bootblock
# lilo -v -C bdlilo.conf -r /mnt/flash
# echo "Setting flags to copy to ramdisk and start the image at block $KERNEL_BI
# Update the kernel image to link to the compressed ramdisk image
# rdev -r /mnt/flash/vmlinuz.embedded 8622
```

```
# Unmount the flash disk
Unmount /mnt/flash
# Write the ramdisk image after the end of the kernel filesystem.
# echo "Writing the compressed root file system image to the floppy after
# the boot image"
dd if=/tmp/ram_image.gz of=/dev/hdc bs=1k seek+$KERNEL_BLOCKS
```

makeCompressedRamDisk script

```
# make Compressed RamDisk script
echo "Making a ram disk device"
# Create a ramdisk by writing zeroes using dd
dd if=/dev/zero of=/dev/ram bs=1k count=4096
# Make a filesystem
echo "Putting an ext2 filesystem on the ram disk"
mke2fs -vm0 /dev/ram 4096
# Mount the ramdisk
echo "Mounting the disk"
mount /dev/ram /mnt/ram
# Copy the root file system to the ramdisk
echo "Copying the root file system"
cp -dpR ./root/* /mnt/ram
# Unmount the ramdisk
echo "Unmounting the disk"
Unmount /mnt/ram
# Compress the ramdisk and copy it to a file
echo "Compressing the image"
dd if=/dev/ram bs=1k count=4096 | gzip -9 > /tmp/ram_image.gz
```

bdlilo.conf script

```
install  =/boot/boot.b
map      =/boot/map
read-write
append="load_ramdisk=1  ramdisk_start=430  ramdisk_size=8192"
backup =/dev/null
compact
image  =vmlinuz.embedded
    label    =  Bootdisk
    root     =  dev/hda
```

CLAIMS

WHAT IS CLAIMED IS:

1. A network flow system comprising:

at least one measurement device;

a flow computer associated with at least one measurement device and configured to receive data from said at least one measurement device, said flow computer also being a web server; and

a host computer connected to said flow computer locally or via an Internet/intranet, wherein said host computer is programmed to transmit and receive data according to a communication protocol and wherein said host computer may communicate with said flow computer by said connection between said host computer and said flow computer.

2. The network flow system of claim 1, wherein said flow computer is identified by a URL address.

3. The network flow system of claim 1, wherein said host computer receives data from said web server in a web page format.

4. The network flow system of claim 1, wherein said communication protocol is an Internet protocol.

5. The network flow system of claim 1, wherein said flow computer is programmable to identify said at least one measurement device without indication from said host computer, and further wherein said flow computer self-configures to communicate with said at least one measurement device.

6. The network flow system of claim 4, wherein said flow computer is programmable to identify said at least one measurement device without indication from said host computer, and further wherein said flow computer self-configures to communicate with said at least one measurement device.

7. The network flow system of claim 1, wherein said flow computer further comprises a storage device and wherein said flow computer is programmed with a scalable operating system recorded on said storage device.

8. The network flow system of claim 7, wherein said operating system is compressed onto said flash disk.

9. The network flow system of claim 1, wherein at least one of said measurement devices communicates with said flow computer by a non-Internet protocol.

10. The network flow system of claim 1, wherein said host computer may remotely program said flow computer via said Internet, or intranet connection.

11. The network flow system of claim 1, wherein said flow computer is security programmed to allow said host computer to program said flow computer only upon the satisfaction of predetermined conditions.

12. The network flow system of claim 1, wherein said host computer communicates with said flow computer via a web browser.

13. A method to obtain measurement data, comprising:

(a) coupling a flow computer to a measurement device, said measurement device suitable to monitor a fluid and provide corresponding measurement data; and

(b) connecting said flow computer via a communication link to a host computer, said flow computer communicating with said measurement device with a first protocol and with said host computer with a second protocol, said second protocol being suitable for Internet/intranet communication wherein said host computer obtains said corresponding measurement data from said measurement device through said flow computer.

14. The method of claim 13, wherein said flow computer queries said measurement device at regular intervals, said measurement device responding with measurement data, wherein said flow computer generates a log of said measurement data.

15. The method of claim 13, wherein said host computer is capable of programming said flow computer.

16. The method of claim 13, wherein said host computer is capable of programming said flow computer using Internet browser software.

17. The method of claim 13, wherein said flow computer is identified by said host computer using a URL address.

18. The method of claim 13, wherein said flow computer is programmed to generate a web page including said corresponding data from said measurement device.

19. A network flow computer, comprising:

a first communication port suited for connection to a measurement device;

a second communication port suited for an Internet/intranet connection;

at least one processor; and

a memory device attached to said at least one processor, said at least one processor programmable to receive data from said measurement device through said first communication port

and programmable to transmit data by said Internet/intranet connection through said second communication port.

20. The network flow computer of claim 19, wherein a compressed operating system is stored on said memory device.

21. The network flow computer of claim 19, wherein a scalable operating system is used in conjunction with said network flow computer.

22. The network flow computer of claim 19, wherein said processor performs calculations on said data received from said measurement device.

23. The network flow computer of claim 22, wherein said processor places a result from said calculations on a web page accessible through said second communication port.

24. The network flow computer of claim 19, wherein said processor places data received from said measurement device on a web page accessible through said second communication port.

FIG. 1

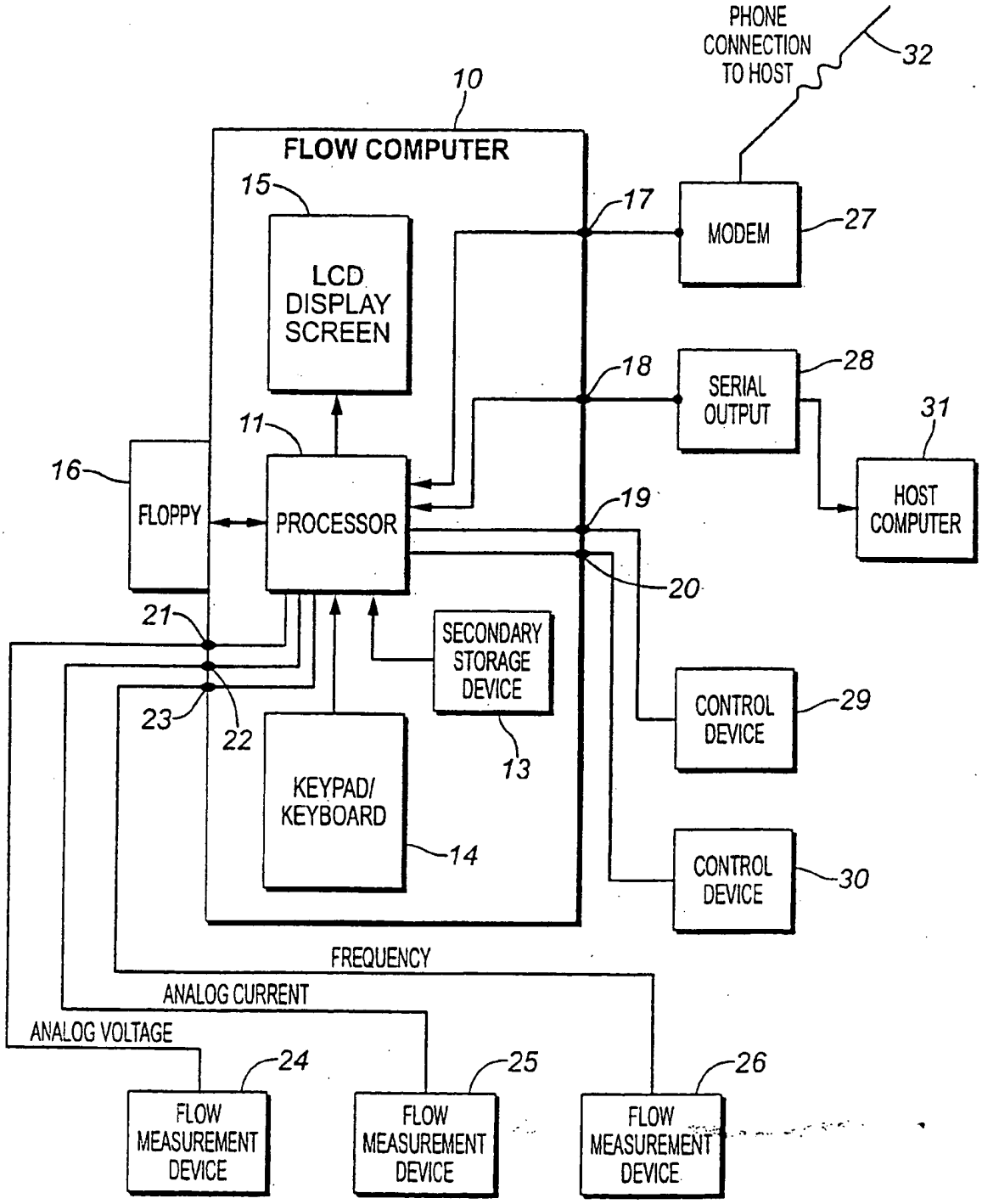

PHONE CONNECTION TO HOST
32
10
FLOW COMPUTER
15
17
MODEM
27
LCD DISPLAY SCREEN
18
SERIAL OUTPUT
28
31
HOST COMPUTER
11
16
FLOPPY
PROCESSOR
19
20
21
SECONDARY STORAGE DEVICE
CONTROL DEVICE
29
23
22
13
KEYPAD/ KEYBOARD
14
CONTROL DEVICE
30
FREQUENCY
ANALOG CURRENT
ANALOG VOLTAGE
24
25
26
FLOW MEASUREMENT DEVICE
FLOW MEASUREMENT DEVICE
FLOW MEASUREMENT DEVICE

FIG. 2

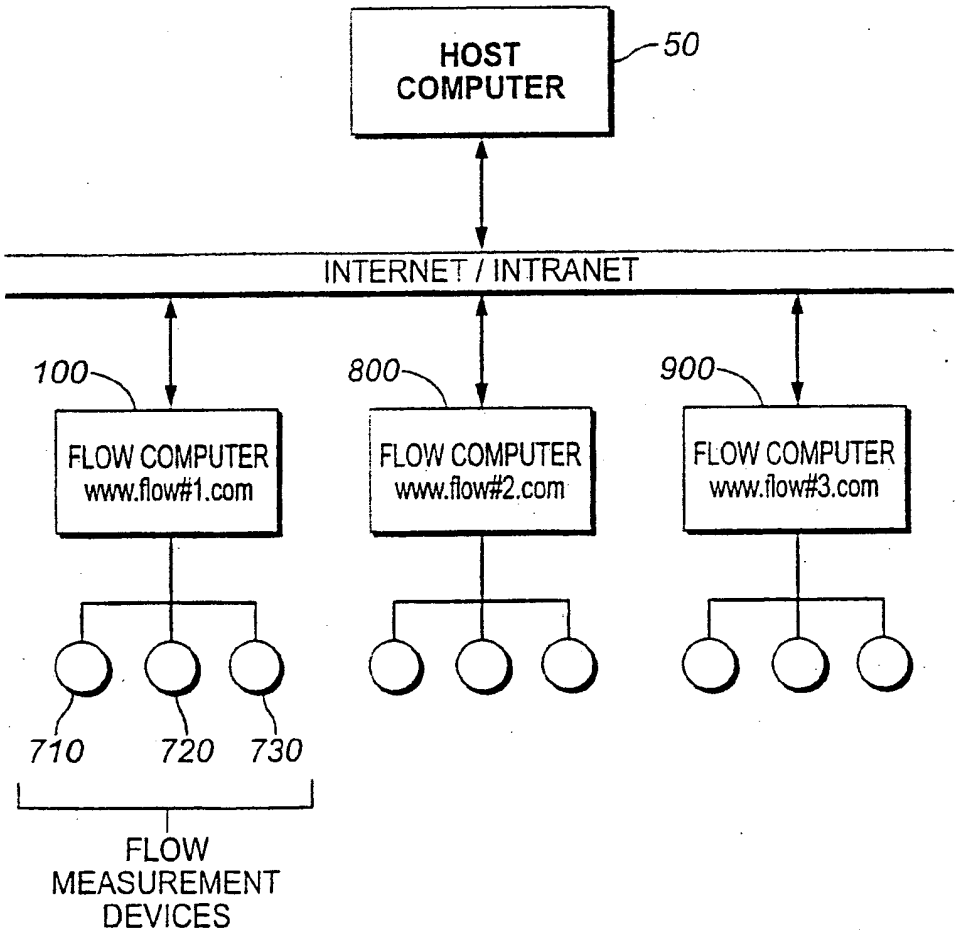

HOST COMPUTER
50
INTERNET / INTRANET
100
800
900
FLOW COMPUTER www.flow#1.com
FLOW COMPUTER www.flow#2.com
FLOW COMPUTER www.flow#3.com
710
720
730
FLOW MEASUREMENT DEVICES

FIG. 3

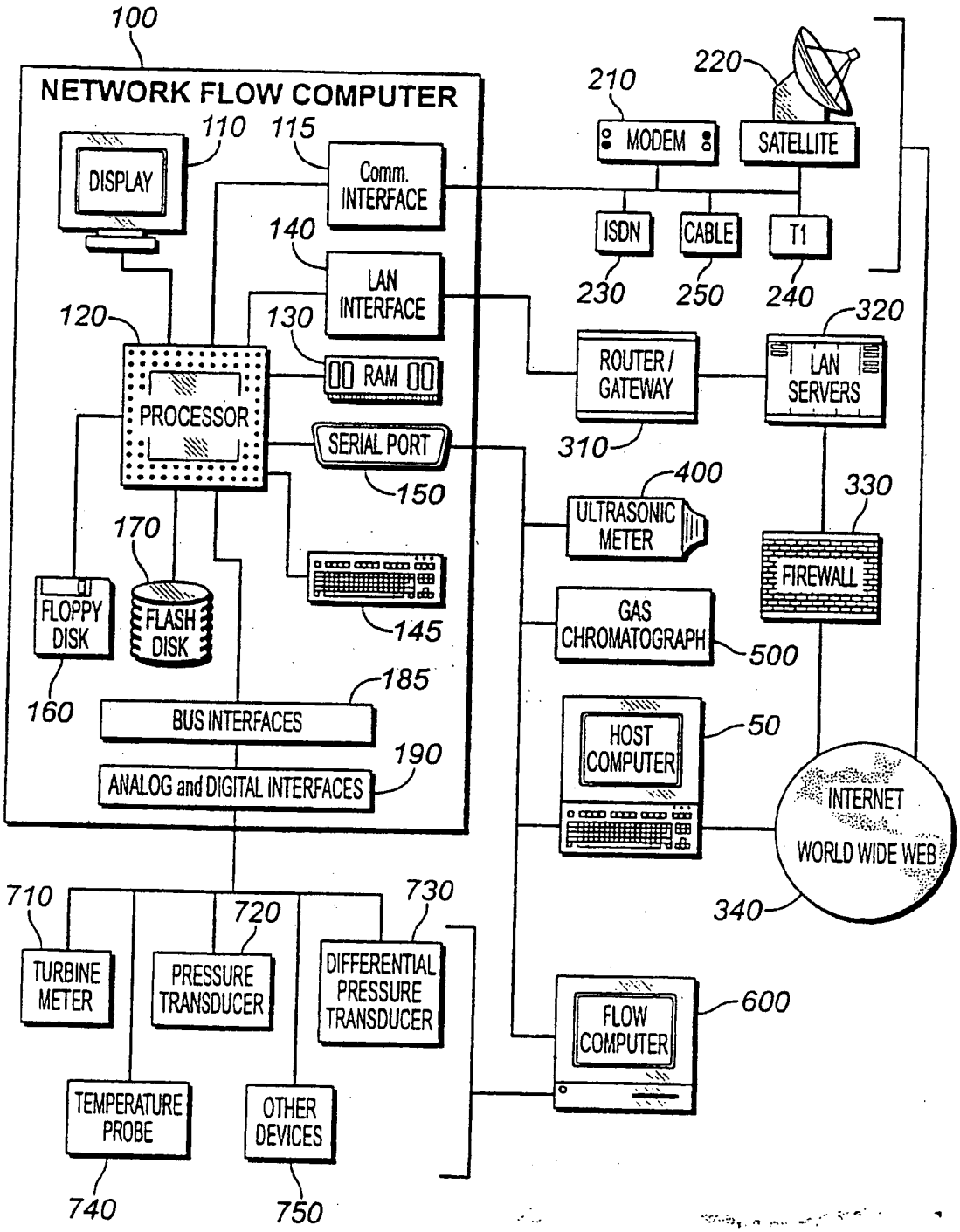

100
NETWORK FLOW COMPUTER
DISPLAY
110
115
Comm. INTERFACE
140
LAN INTERFACE
120
130
RAM
PROCESSOR
SERIAL PORT
150
170
FLOPPY DISK
FLASH DISK
145
160
185
BUS INTERFACES
190
ANALOG and DIGITAL INTERFACES
210
MODEM
220
SATELLITE
ISDN
CABLE
T1
230
250
240
320
ROUTER / GATEWAY
LAN SERVERS
310
400
ULTRASONIC METER
330
FIREWALL
GAS CHROMATOGRAPH
500
HOST COMPUTER
50
INTERNET
WORLD WIDE WEB
340
710
720
730
TURBINE METER
PRESSURE TRANSDUCER
DIFFERENTIAL PRESSURE TRANSDUCER
FLOW COMPUTER
600
TEMPERATURE PROBE
OTHER DEVICES
740
750

FIG. 4

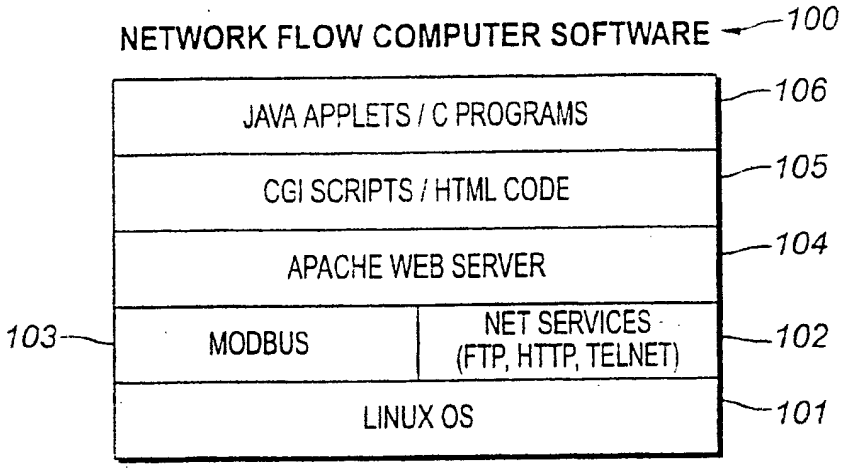

NETWORK FLOW COMPUTER SOFTWARE
100
JAVA APPLETS / C PROGRAMS
106
CGI SCRIPTS / HTML CODE
105
APACHE WEB SERVER
104
103
MODBUS
NET SERVICES (FTP, HTTP, TELNET)
102
LINUX OS
101

FIG. 5

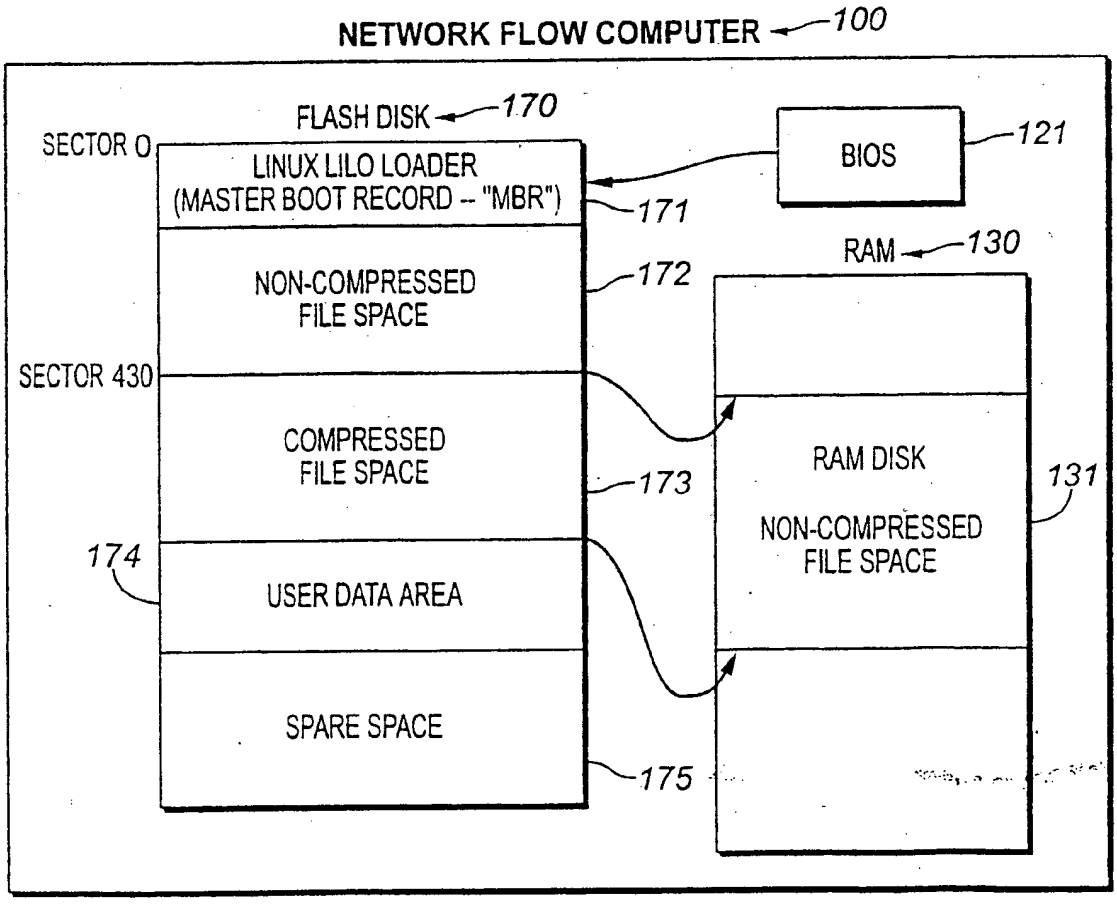
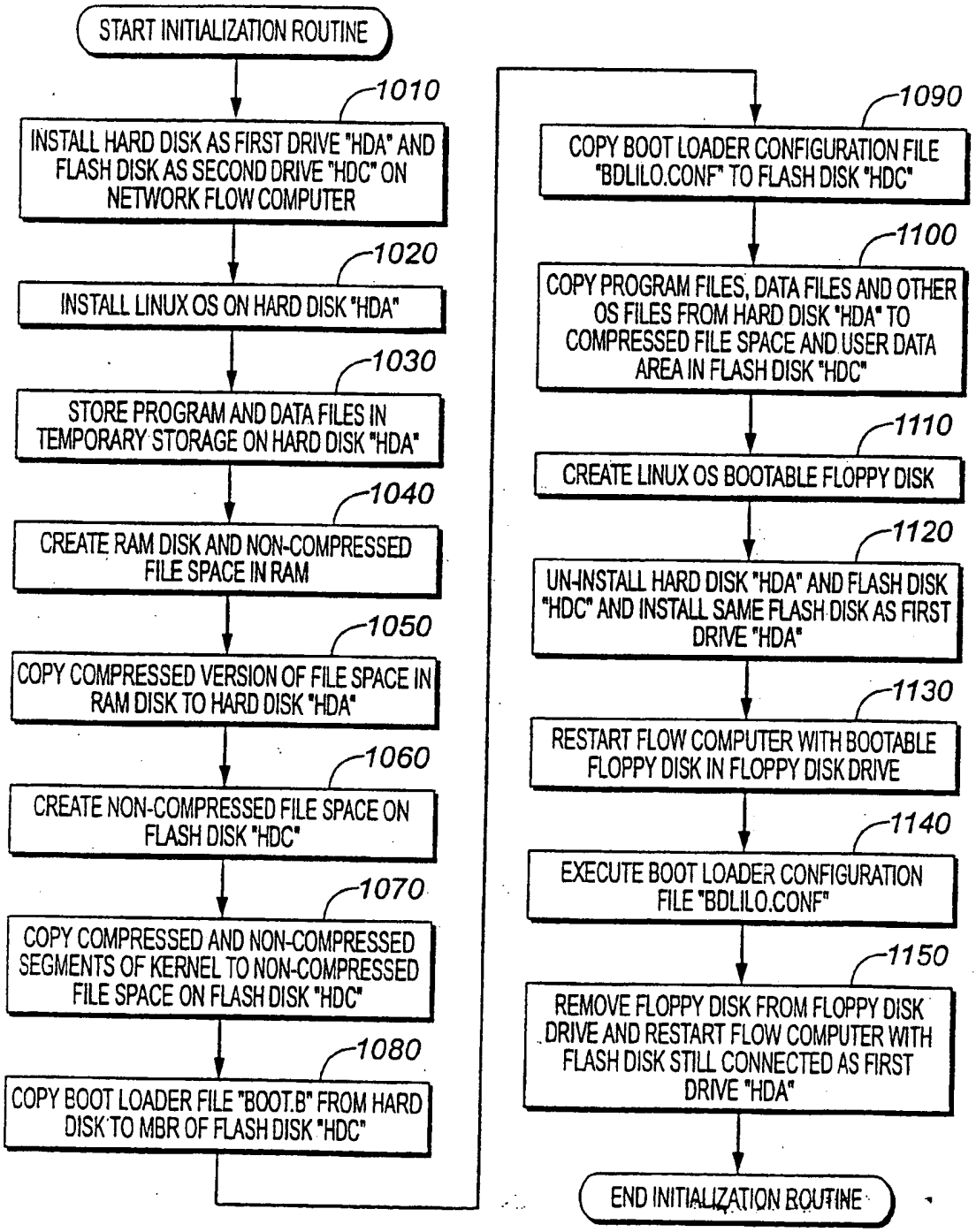

NETWORK FLOW COMPUTER
100
FLASH DISK
170
SECTOR 0
LINUX LILO LOADER (MASTER BOOT RECORD -- "MBR")
171
BIOS
121
NON-COMPRESSED FILE SPACE
172
RAM
130
SECTOR 430
COMPRESSED FILE SPACE
173
RAM DISK
NON-COMPRESSED FILE SPACE
131
174
USER DATA AREA
SPARE SPACE
175

FIG. 6


START INITIALIZATION ROUTINE
1010
INSTALL HARD DISK AS FIRST DRIVE "HDA" AND FLASH DISK AS SECOND DRIVE "HDC" ON NETWORK FLOW COMPUTER
1020
INSTALL LINUX OS ON HARD DISK "HDA"
1030
STORE PROGRAM AND DATA FILES IN TEMPORARY STORAGE ON HARD DISK "HDA"
1040
CREATE RAM DISK AND NON-COMPRESSED FILE SPACE IN RAM
1050
COPY COMPRESSED VERSION OF FILE SPACE IN RAM DISK TO HARD DISK "HDA"
1060
CREATE NON-COMPRESSED FILE SPACE ON FLASH DISK "HDC"
1070
COPY COMPRESSED AND NON-COMPRESSED SEGMENTS OF KERNEL TO NON-COMPRESSED FILE SPACE ON FLASH DISK "HDC"
1080
COPY BOOT LOADER FILE "BOOT.B" FROM HARD DISK TO MBR OF FLASH DISK "HDC"
1090
COPY BOOT LOADER CONFIGURATION FILE "BDLILO.CONF" TO FLASH DISK "HDC"
1100
COPY PROGRAM FILES, DATA FILES AND OTHER OS FILES FROM HARD DISK "HDA" TO COMPRESSED FILE SPACE AND USER DATA AREA IN FLASH DISK "HDC"
1110
CREATE LINUX OS BOOTABLE FLOPPY DISK
1120
UN-INSTALL HARD DISK "HDA" AND FLASH DISK "HDC" AND INSTALL SAME FLASH DISK AS FIRST DRIVE "HDA"
1130
RESTART FLOW COMPUTER WITH BOOTABLE FLOPPY DISK IN FLOPPY DISK DRIVE
1140
EXECUTE BOOT LOADER CONFIGURATION FILE "BDLILO.CONF"
1150
REMOVE FLOPPY DISK FROM FLOPPY DISK DRIVE AND RESTART FLOW COMPUTER WITH FLASH DISK STILL CONNECTED AS FIRST DRIVE "HDA"
END INITIALIZATION ROUTINE

INTERNATIONAL SEARCH REPORT

Internatic Application No
PCT/US 99/29830

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G01F15/06

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G01F G06F H04Q

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	DE 298 14 521 U (SIEMENS AG) 22 October 1998 (1998-10-22) page 3; figure 1	1,13,19
A	US 5 767 790 A (JOVELLANA BARTOLOME D) 16 June 1998 (1998-06-16) abstract; figures 1,2	1,13,19
A	US 5 808 558 A (MEEK JEAN L ET AL) 15 September 1998 (1998-09-15) abstract; figure 1	1,13,19

[ ] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
1 March 2000	09/03/2000

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31–70) 340–2040, Tx. 31 651 epo nl, Fax: (+31–70) 340–3016	Vorropoulos, G

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

Internatic Application No
PCT/US 99/29830

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
DE 29814521 U	22-10-1998	NONE	
US 5767790 A	16-06-1998	NONE	
US 5808558 A	15-09-1998	US 5602744 A	11-02-1997